User's Manual

M16C Series,R8C Family
C Compiler Package V.5.45
Assembler User's Manual

Renesas Electronics
www.renesas.com

Rev.2.00 2010.04

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

AS30 Contents

AS30 Contents 4
Manual Writing Conventions 9
Specifications of AS30 10
Character Set 10
Outline of Function 11
Configuration 11
Functions 12
Outline Processing by AS30 13
Structure of as30 14
Outline of as30 functions 16
Outline of ln30 functions 17
Outline of lmc30 functions 18
Outline of lb30 functions 19
Outline of xrf30 functions 20
Outline of abs30 functions 21
AS30 Functions 22
Relocatable Assemble 22
Unit of Address Management (Section) 22
Rules on Section Management 24
Label and symbol 26
Management of Label and Symbol Addresses 27
Library File Referencing Function 28
Management of Include File 30
Code Selection by AS30 31
Optimized Code Selection 31
Outline of Mnemonic Description 31
Optimized Selection by AS30 31
SB Register Offset Description 34
Special Page Branch 34
Special Page Subroutine 34
Special Page Vector Table 35
Macro Function 36
Conditional Assemble Control 39
Structured Description Function 40
Source Line Information Output 40
Symbol Definition 40
Environment Variables of as30 41
Output messages 43
Input/Output Files of AS30 44
Relocatable Module File 44
Assembler List File 45
Assembler Error Tag File 48
Branch Information File 48
Absolute Module File 48
Map File 49
Link Error Tag File 51
Motorola S Format 51
Intel HEX Format 51
ID File 52
Library File 52
Library List File 52
Cross Reference File 53
Absolute List File 54
Starting Up Program 55

Precautions on Entering Commands....55
Structure of Command Line....55
Rules for Entering Command Line....56
Method for Operating as30....57
Command Parameters....57
Rules for Specifying Command Parameters....58
Include File Search Directory....58
as30 Command Options....59
-....59
-A....59
-C....60
-D....61
-finfo....61
-F....62
-H....63
-I....63
-JOPT....63
-L....64
-M....64
-M60/-M61....65
-N....65
-O....66
-P....66
-PATCH(6N)_TA/-PATCH(6N)_TAn....67
-R8C/-R8CE....68
-R8Cxx....69
-S....70
-T....70
-V....71
-X....71
Error Messages of as30....72
Warning Messages of as30....79
Method for Operating ln30....81
Command Parameters....81
Rules for Specifying Command Parameters....82
Command File....83
Command Options of ln30....84
-....84
-E....84
-G....85
-JOPT....85
-L....86
-LD....87
-LOC....88
-M....89
-M60/-M61....89
-MS/-MSL....89
-NOSTOP....90
-O....90
-ORDER....91
-R8C/-R8CE....92
-T....92
-U....93
-V....93
-VECT....94
-VECTN....95
-W....96

@ 96
Error Messages of ln30 97
Warning Messages of ln30 99
Method for Operating lmc30 102
Command Parameters 102
Rules for Specifying Command Parameters 102
lmc30 Command Options 103
-. 103
-A 103
-E 104
-F 105
-H 106
-ID 107
-L 108
-O 108
-ofsregx 109
-protect1 109
-protectx 110
-R8C/-R8C 111
-V 111
Error Messages of lmc30 112
Warning Messages of lmc30 113
Method for Operating lb30 114
Command Parameters 114
Rules for Specifying Command Parameters 114
Command Options of lb30 116
-. 116
-A 116
-C 117
-D 117
-L 118
-R 118
-U 119
-V 119
-X 119
@ 120
Error Messages of lb30 121
Warning Messages of lb30 122
Method for Operating xrf30 123
Command Parameters 123
Rules for Specifying Command Parameters 123
Command Options of xrf30 124
-. 124
-N 124
-O 124
-V 125
@ 125
Error Messages of xrf30 126
Method for Operating abs30 127
Precautions using abs30 127
Command Parameters 127
Rules for Specifying Command Parameters 127
Command Options of abs30 127
-. 128
-D 128
-O 128
-V 129

Error Messages of abs30 130
Warning Messages of abs30 130
Rules for Writing Program 131
Precautions on Writing Program 131
Character Set 131
Reserved Words 131
Names 132
Lines 134
Line concatenation 138
Operands 138
Operators 140
Character String 142
Directive Commands 143
List of Directive Commands 144
..FILE 148
..MACPARA 149
..MACREP 150
.ADDR 151
.ALIGN 152
.ASSERT 153
.BLKA 154
.BLKB 155
.BLKD 156
.BLKF 157
.BLKL 158
.BLKW 159
.BTEQU 160
.BTGLB 161
.BYTE 162
.CALL 163
.DEFINE 164
.DOUBLE 165
.EINSF 166
.ELIF 167
.ELSE 168
.END 169
.ENDIF 170
.ENDM 171
.ENDR 172
.EQU 173
.EXITM 174
.FB 175
.FBSYM 176
.FLOAT 177
.FORM 178
.GLB 179
.ID 180
.IF 181
.INCLUDE 183
.INITSCT 184
.INSF 185
.INSTR 186
.LEN 187
.LIST 188
.LOCAL 189
.LWORD 190
.MACRO 191

.MREPEAT 193
.OFSREG 194
.OPTJ 195
.ORG 196
.PAGE 198
.PROTECT 199
.RVECTOR 200
.SB 201
.SBBIT 202
.SBSYM 203
.SB_AUTO 204
.SECTION 205
.SJMP 206
.STK 207
.SUBSTR 208
.SVECTOR 209
.VER 210
.WORD 211
? 212
@ 213
Structured Description Function 214
Outline 214
Structured Description Statement 214
Reserved Variables 216
Memory Variables 218
Memory Bit Variables 221
Structured Operators 222
Structure of Structured Description Statement 224
List of Structured Description Commands 225
IF Statement 226
FOR-STEP Statement 228
FOR-NEXT Statement 230
SWITCH Statement 231
DO Statement 233
BREAK Statement 234
CONTINUE Statement 235
FOREVER Statement 236
Assignment Statement 237
Structure of Structured Description Commands 240
Syntax of Statements 242

Manual Writing Conventions

The following explains the conventions used in writing AS30 User's Manuals and the discrimination between uppercase (capital) and lowercase (small) letters used in these manuals.

Uppercase English letters (A to Z)

Indicate the character strings such as mnemonics, directive commands, and reserved words you may write in source programs and command lines without modifying the written character strings.

Lowercase English letters

Indicate the character strings you can replace with your desired character string. For example, these indicate label names you can enter as you want. However, the AS30 program names and file extensions cannot be replaced with other character strings.

\

Indicates separation between directories. The AS30 manuals use the MS-DOS notation to show command input examples unless otherwise noted.

[ ]

Indicates that descriptions in [ ] can be omitted.

[a|B]

Indicates that you can select one of two items separated by | in [ ] as you write a program.
Terms Used in Manuals
The following explains the terms used in AS30 User's Manuals.

AS30

Collectively refers to the programs included in the AS30 system or denotes the AS30 software package for the M16C family.

as30, mac30, pre30, asp30, ln30, lmc30, lb30, xrf30, abs30

Refer to the executable program names included with AS30. Program names are written in lowercase letters unless otherwise noted.

AS30.EXE, MAC30.EXE, PRE30.EXE, ASP30.EXE, LN30.EXE, LMC30.EXE, LB30.EXE, XRF30.EXE, ABS30.EXE

Refer to the execution program names on MS-DOS.

Mnemonic

Refers to assembly language instructions for the M16C family.

Instructions

Collectively refer to the mnemonics and AS30 directive commands.

Source program

Refers to program descriptions that can be processed by AS30.

Assembly source file

Refers to files that contain a source program.

Specifications of AS30

AS30 has been designed based on the following specifications. Make sure AS30 in your system is used within the range of this specification.

Item	Specification
Number of files opened simultaneously	Maximum 9 files
Number of characters that can be set in environment variables	2048 bytes (characters)
Number of characters per line of source file	512 bytes (characters)
Number of macro definitions	65535

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 s 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)

Precaution

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.

Outline of Function

AS30 is a software system that assists you at the assembly language level in developing control programs for the M16C family of single-chip microcomputers.

It converts source programs written in assembly language into files of source level debuggable format. AS30 also includes a program that converts source programs into files of M16C family ROM programmable format. Furthermore, AS30 can be used in combination with the C compiler (NC30).

Configuration

AS30 consists of the following programs.

Assembler driver (as30)

This program starts up macro processor ,structured 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.

Precaution

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.

Structured processor

This program processes structured directive commands in the assembly source and generates an assembly source file.

Assembler processor

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

Linkage editor (ln30)

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

Librarian (lb30)

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

Load module converter (lmc30)

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 (xrf30)

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 (abs30)

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

AS30 has a function to select the addressing mode and branch instruction that are most efficient in generating code for the source program.

Macro function

AS30 has a macro function to improve a program's readability.
Source level debug information output in high-level language
AS30 outputs source level debuggable high-level language information for programs developed in M16C family's high-level languages.

File generation

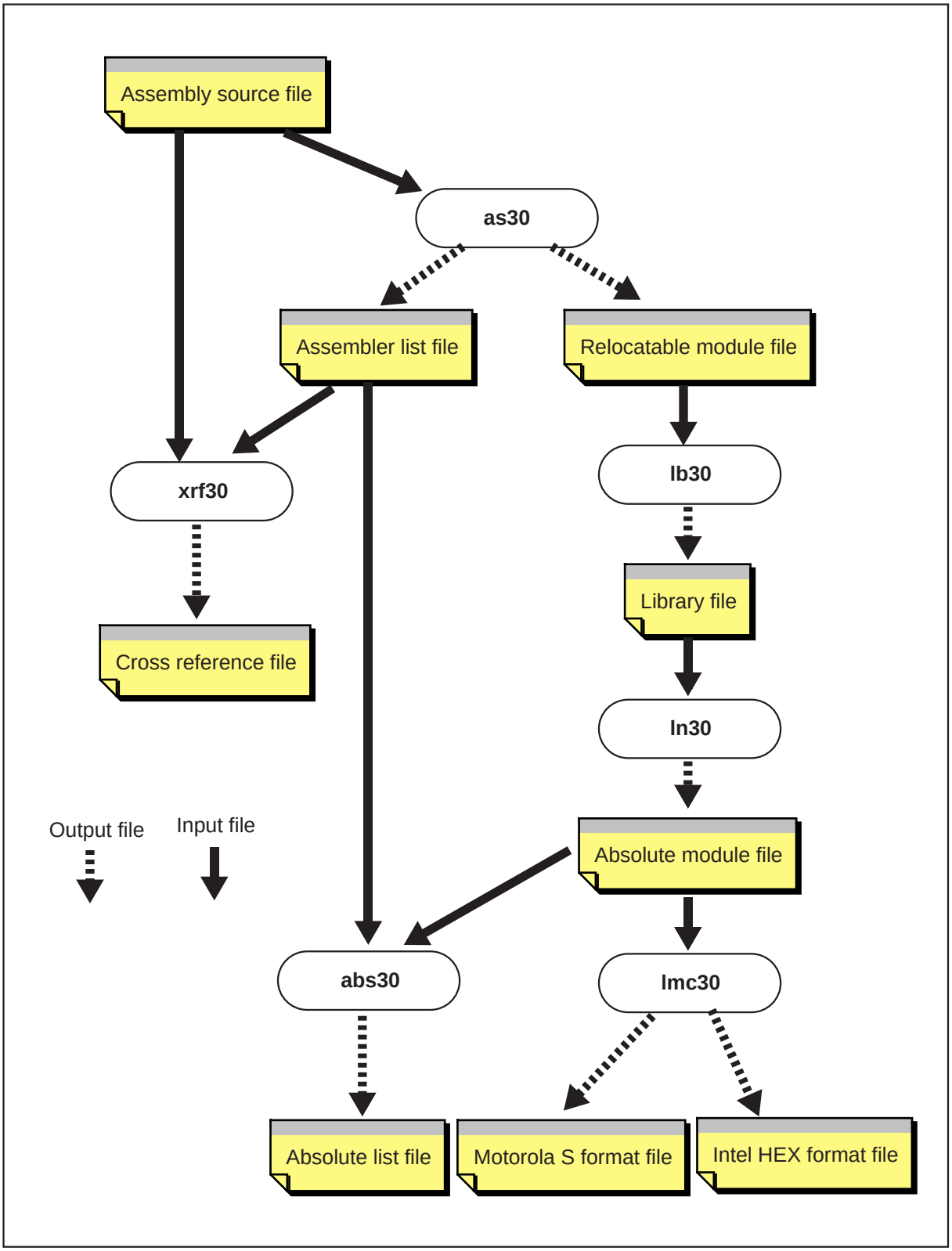
Each program in AS30 generates a relocatable module file, absolute module file, error tag file, list file, and others.

IEEE-695 format file generating function

The binary files generated by AS30 are output in IEEE- 695 format. Therefore, AS30 can be shared with other M16C family development tools using formats based on the IEEE-695 format.
IEEE (Institute of Electrical and Electronics Engineers, USA)

Outline Processing by AS30

The diagram below schematically shows assemble processing by AS30.



Structure of as30

The as30 assembler consists of a program to processing macro descriptions a program to processing structured descriptions and a program to convert an assembly source file into an relocatable module file. The name as30 represents a program to control these two programs.

Precaution
AS30 uses the assembler driver to control the macro processor structured processor and assembler processor. Therefore, neither macro processor structured processor nor assembler processor can be invoked directly from your command line. Program operation is not guaranteed if the macro processor, structured processor or assembler processor is invoked directly.

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 structured processor or assembler processor.

Outline of structured processor functions

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

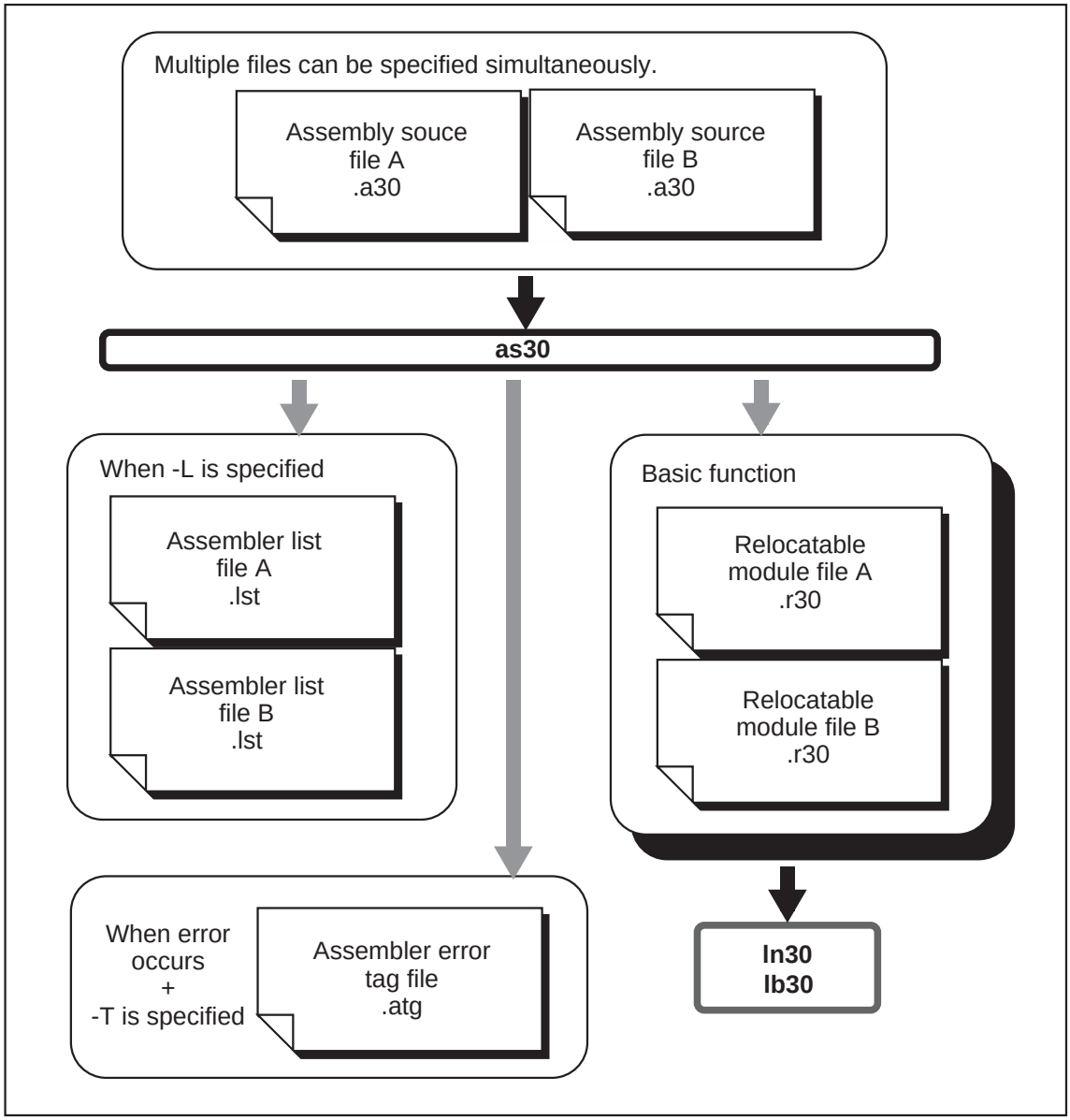
Outline of asp30 functions

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

Outline Processing by as30

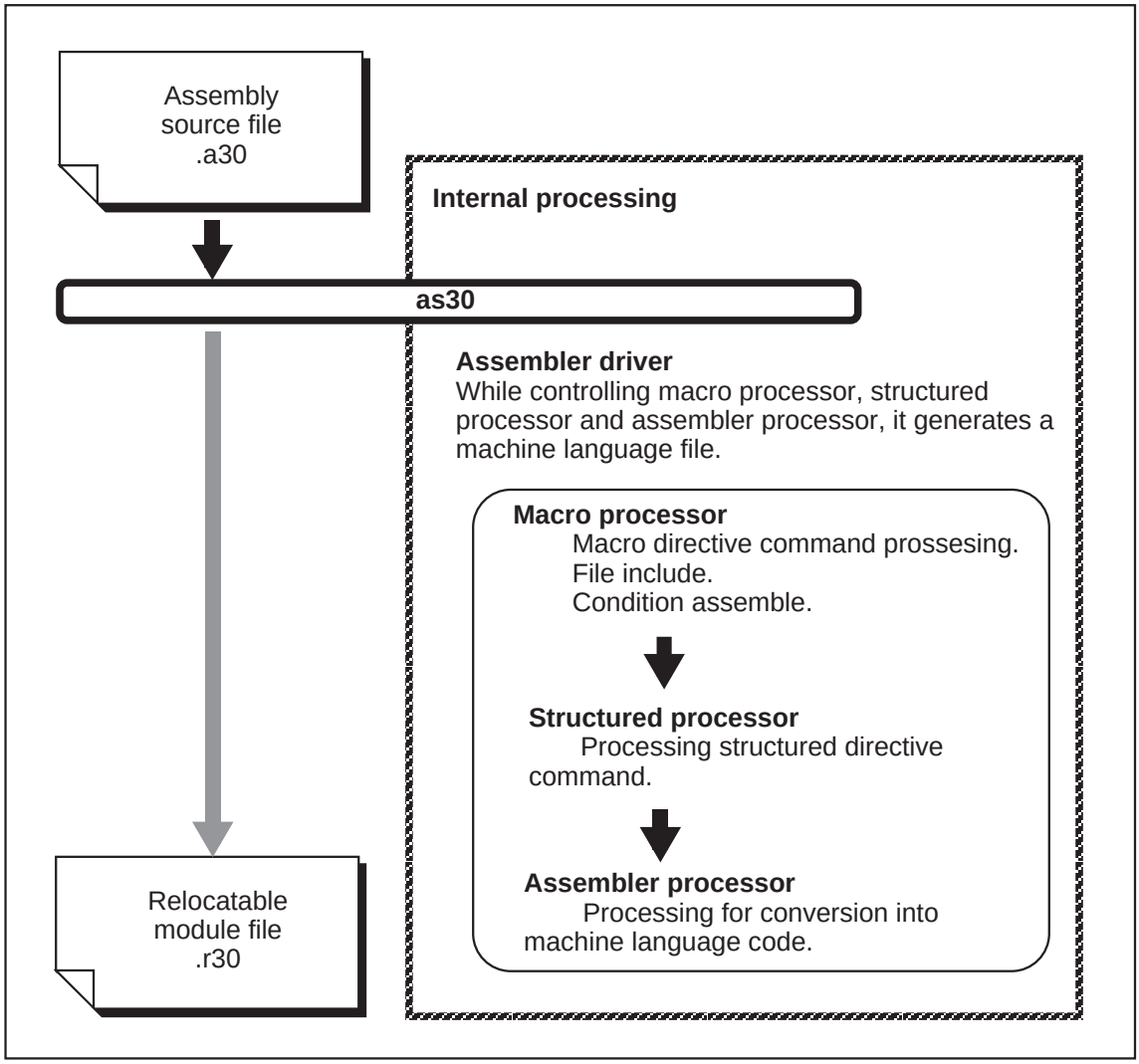
- After interpreting the input command lines, as30 activates each program of macroprocessor, structured processor, and assembler processor.
- as30 controls the command options added when each program starts up and the file names to be processed.
- Each program is started up sequentially in the following order:
  1 Macro processor
  2 Structured processor
  3 Assembler processor

The chart below shows a flow of processing performed by as30.



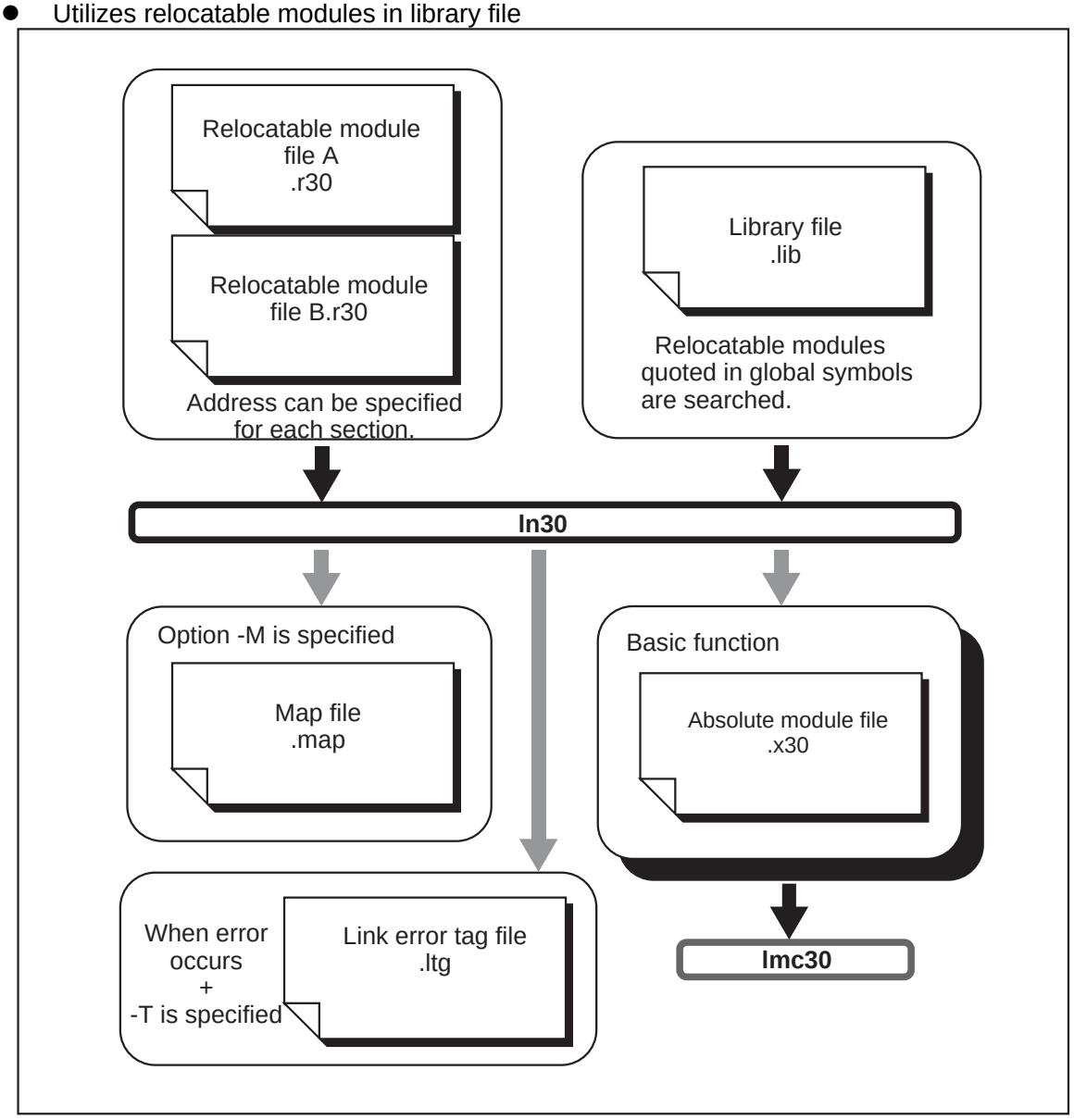
Outline of as30 functions

- Generates relocatable module files
- Generates assembler list files



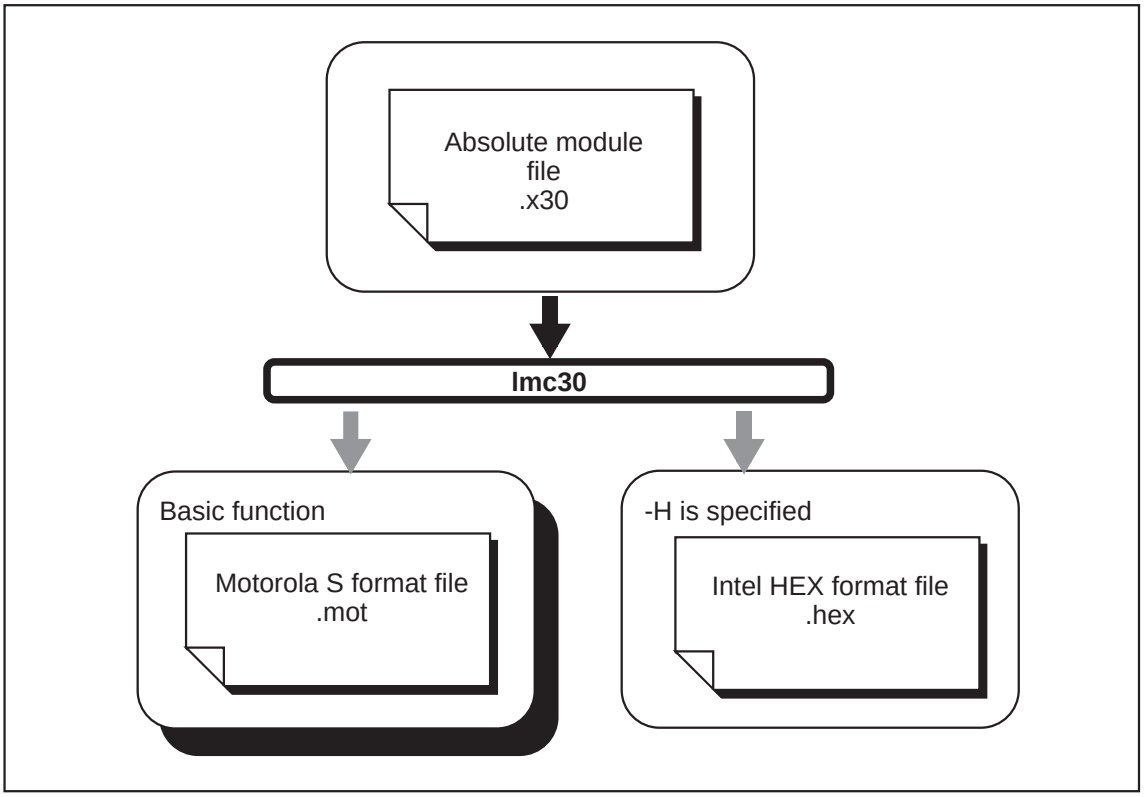
Outline of ln30 functions

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



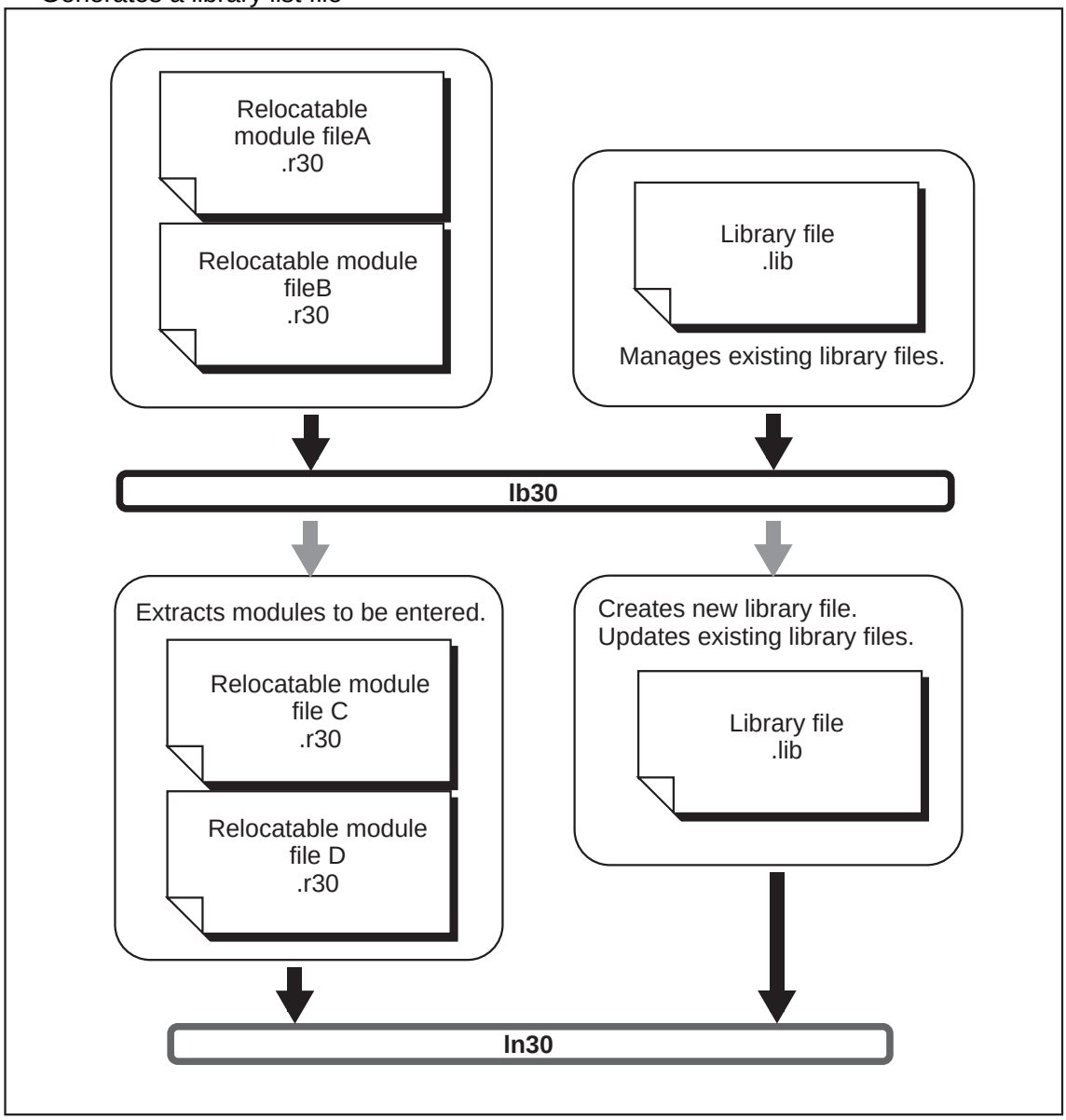
Outline of lmc30 functions

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



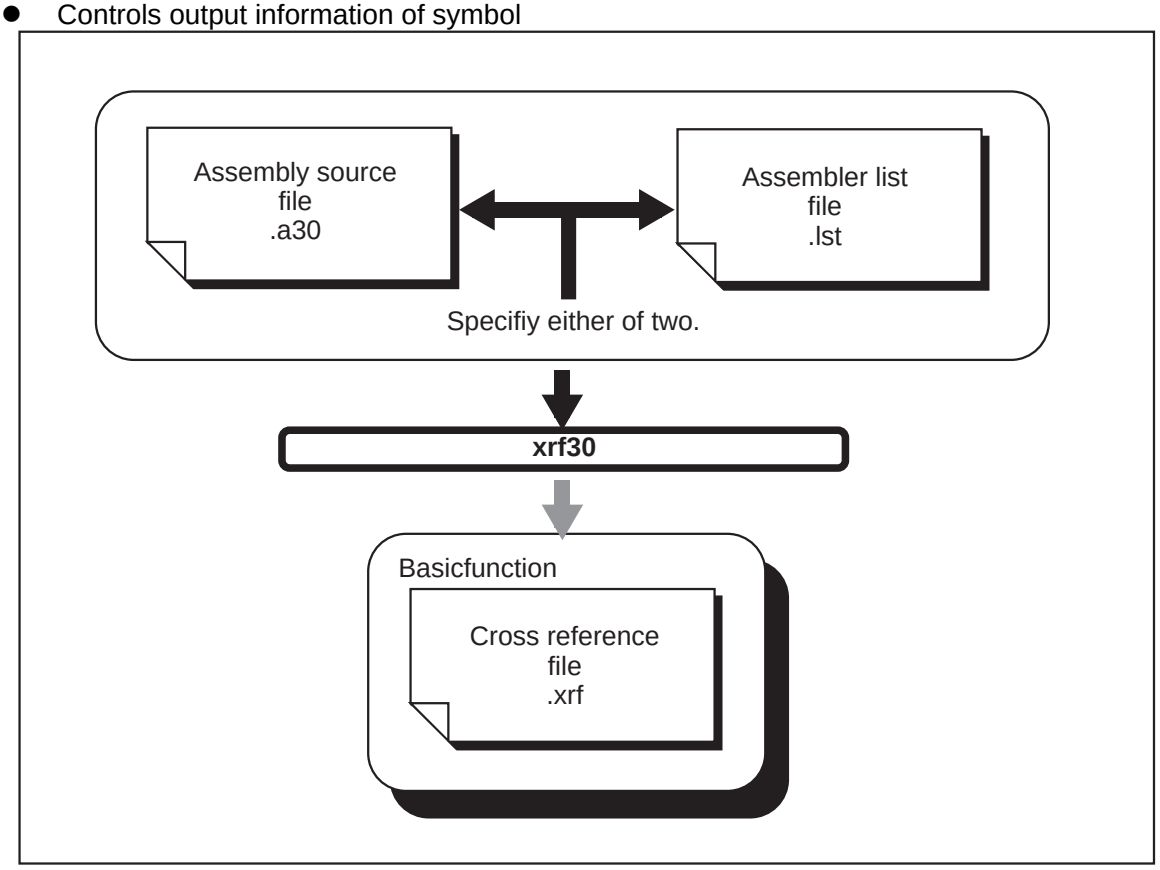
Outline of lb30 functions

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



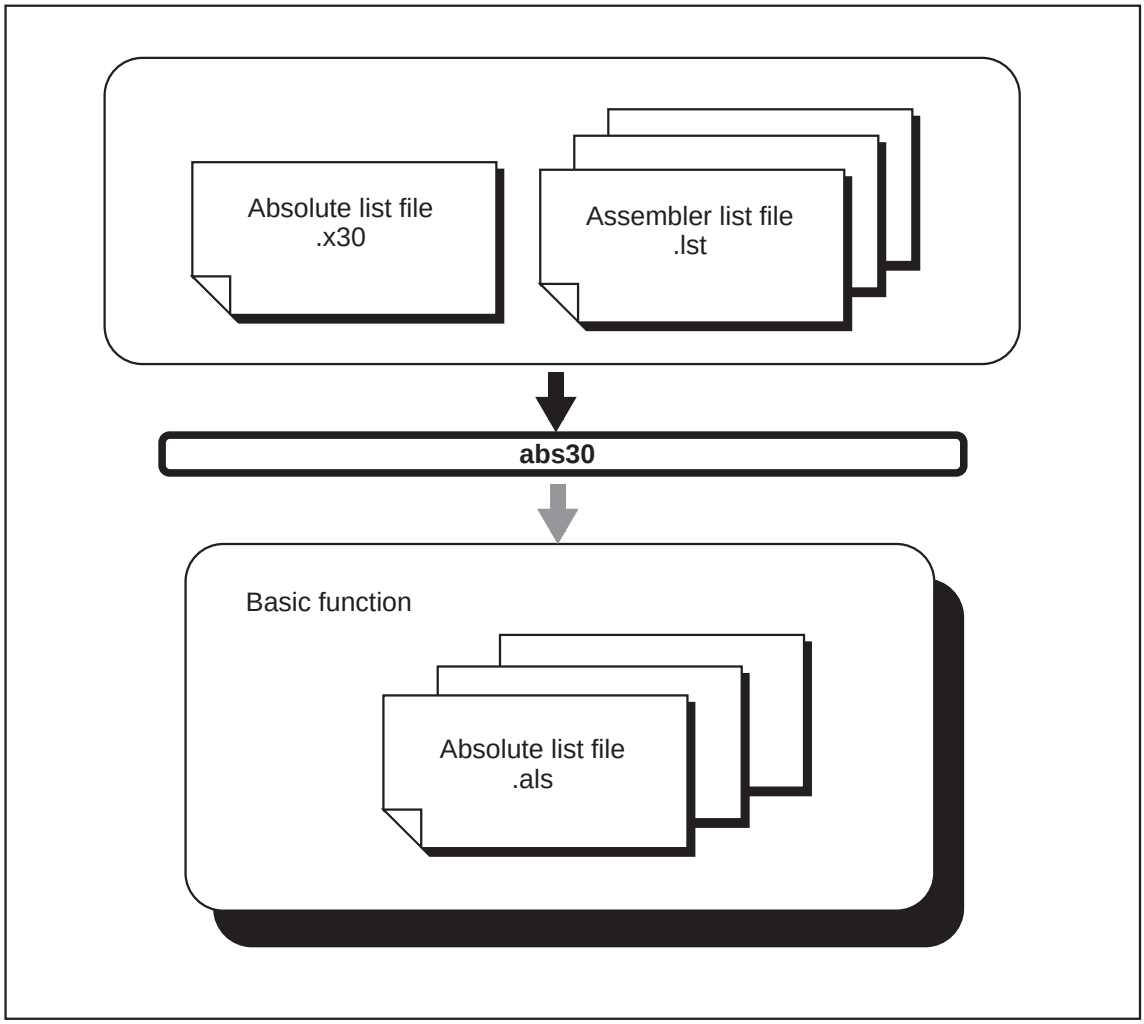
Outline of xrf30 functions

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



Outline of abs30 functions

- Generates absolute list files



AS30 Functions

The as30 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 as30 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 ln30 editor references section information and global symbol information in the relocatable module file generated by as30 while it determines the addresses for the absolute module file section by section.

Precautions

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 as30 and ln30 have no concern for the physical address locations in the actual ROM and RAM of each microcomputer in the M16C family. 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)

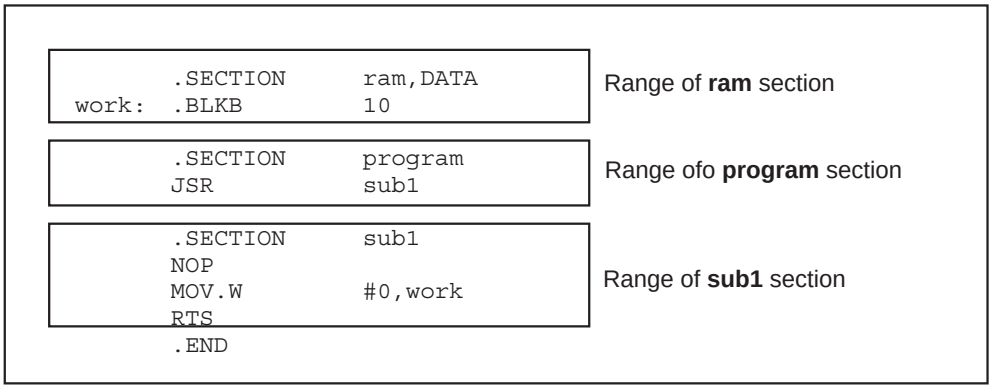
AS30 manages addresses in units of sections.

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.

Precautions

Sections cannot be nested by definition.



Type of Section

You can set a type for a section in which units addresses are controlled by AS30. 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 AS30 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:

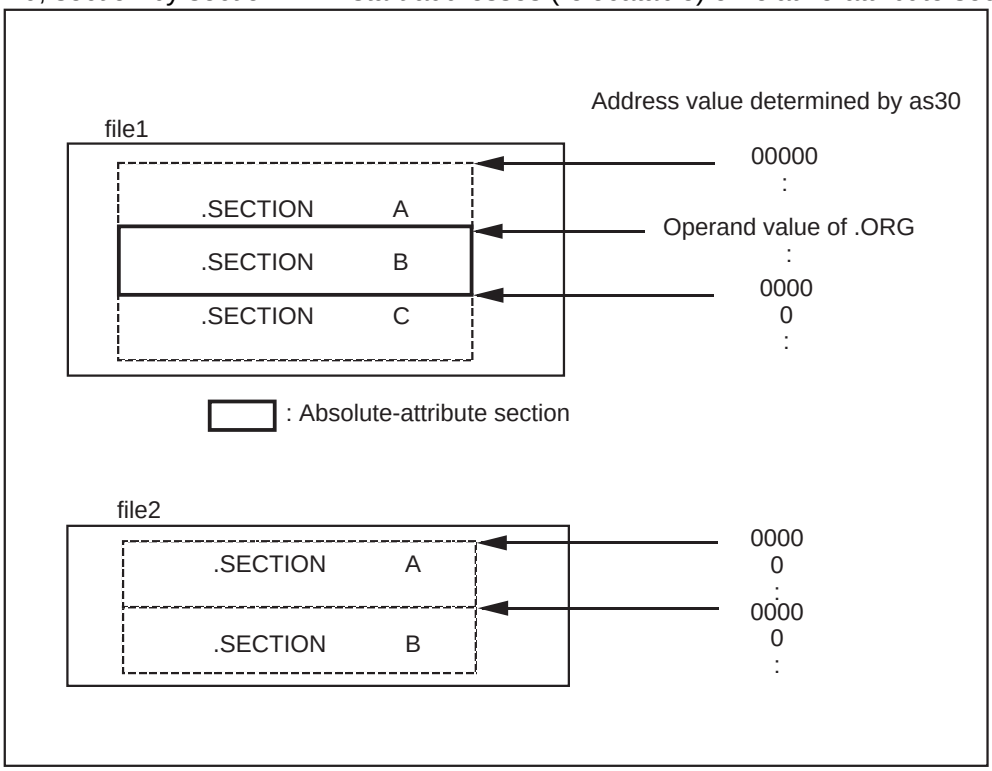
```
.SECTION    program,CODE
.ORG        1000H
```

Rules on Section Management

This section describes how AS30 converts the source program written in multiple files into a single executable file.

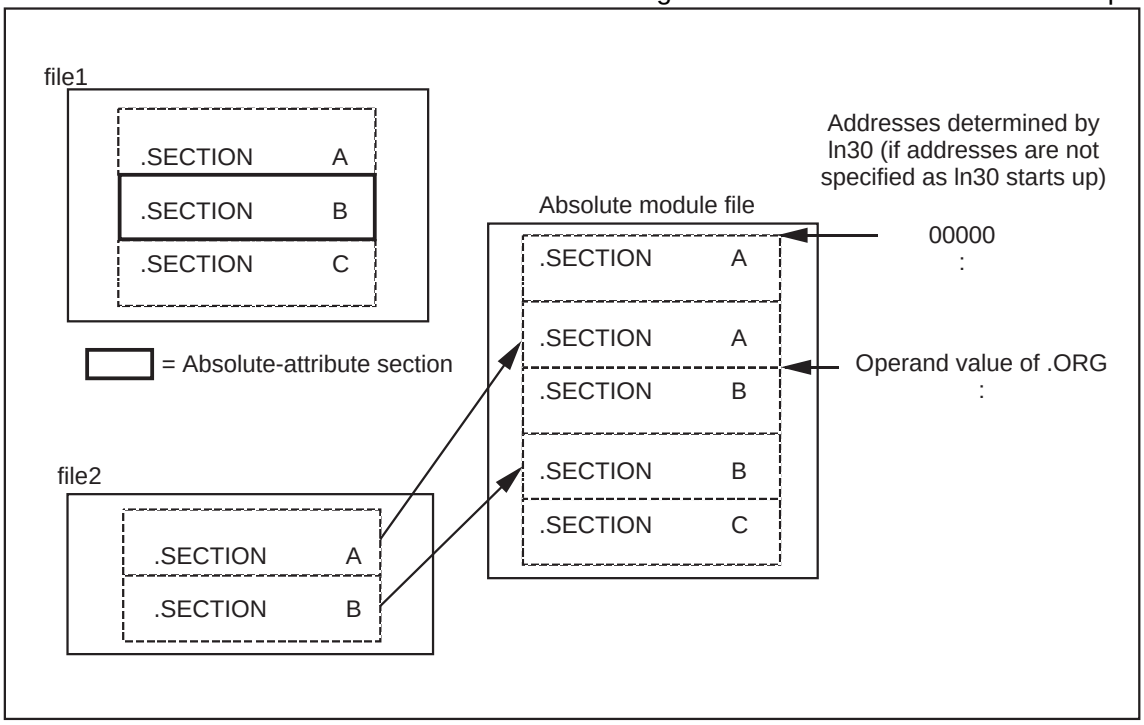
Section Management by as30

- 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 ln30

- 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.



Rules for section allocation by ln30

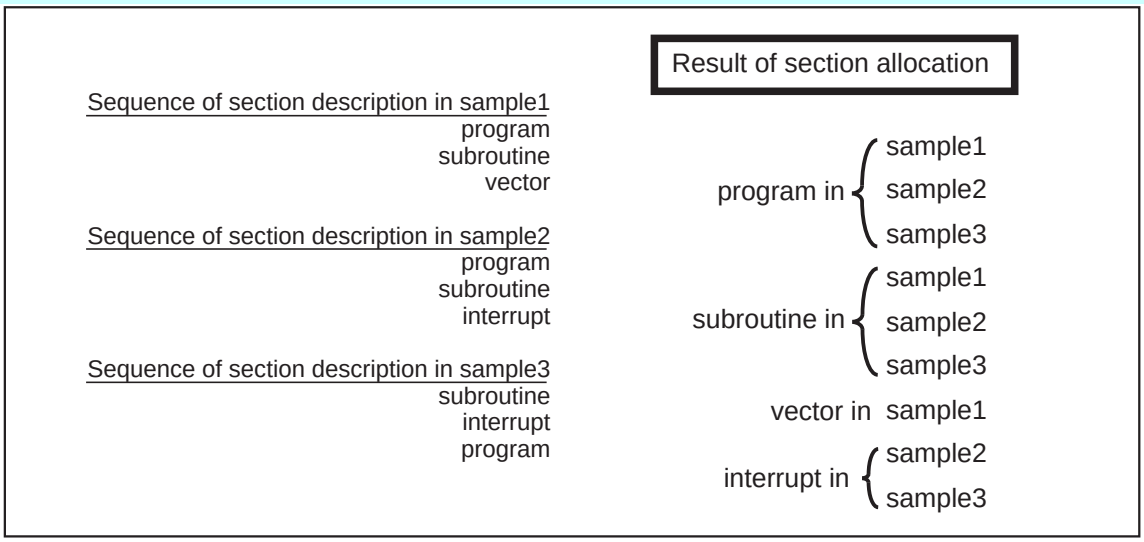
- Sections of the same name are allocated in order of files specified.
- Sections are allocated in the order they are entered in the file that is specified first.
- If an attempt is made to allocate an absolute attribute after another absolute attribute, ln30 outputs a warning.
- For section type "DATA", if addresses overlap in two or more sections, ln30 outputs a warning. Sections will be allocated overlapping each other.
- For section type "CODE" or "ROMDATA", if addresses overlap in two or more sections, ln30 outputs an error.
- If an attempt is made to allocate an absolute attribute after a relative attribute in sections of the same name, ln30 outputs an error.
- If section names are the same and information on section attribute or type is inconsistent, ln30 outputs a warning.

Example of section allocation by ln30

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

Example: Three relocatable module files are linked by entering the command below. In this case, the generated absolute module file is named "sample.x30".

```
>ln30 sample1 sample2 sample3
```



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
```

Label and symbol

The as30 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 AS30.

AS30 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.

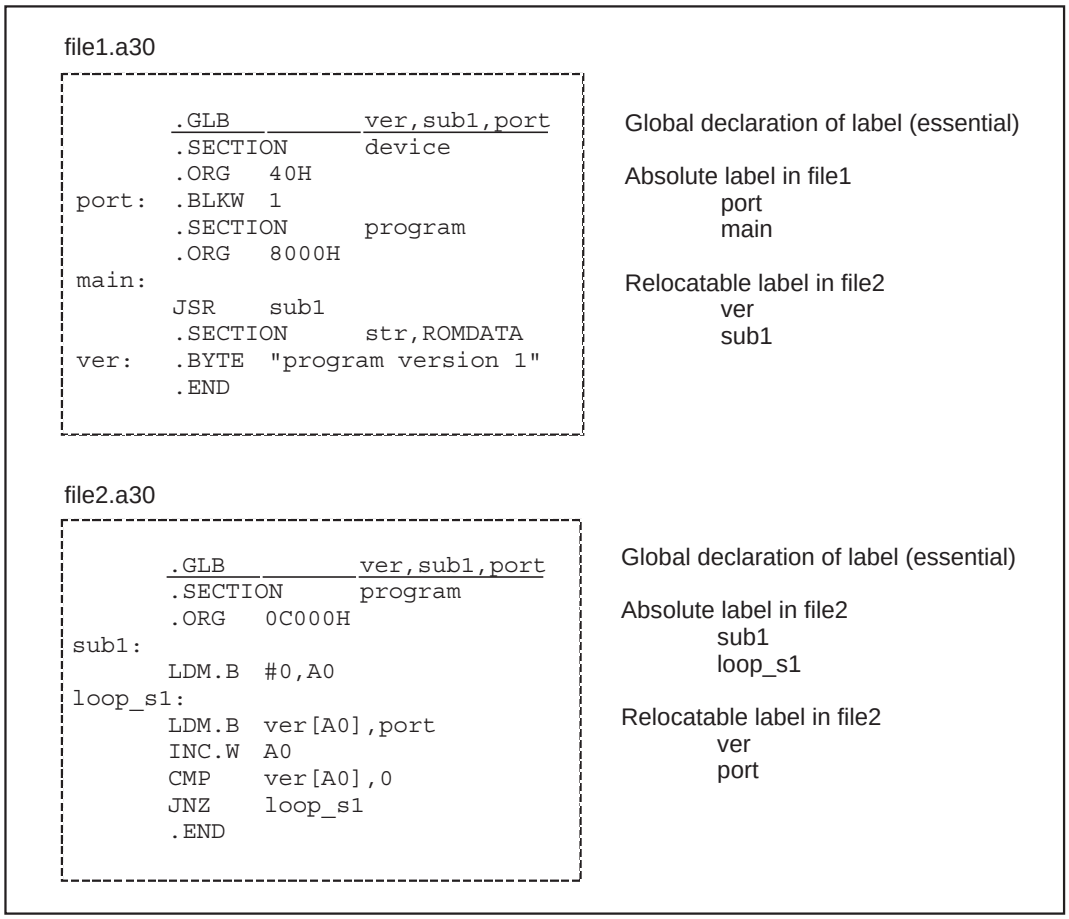
Relocatable

- 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 ln30 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, ln30 outputs a warning.

If the determined actual address lies outside the range of branch instructions and addressing modes determined by as30.

Library File Referencing Function

If all of the following conditions are met, ln30 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.

Precaution

The ln30 editor links the entire relocatable module where necessary global labels are defined.

Rules for referencing library modules

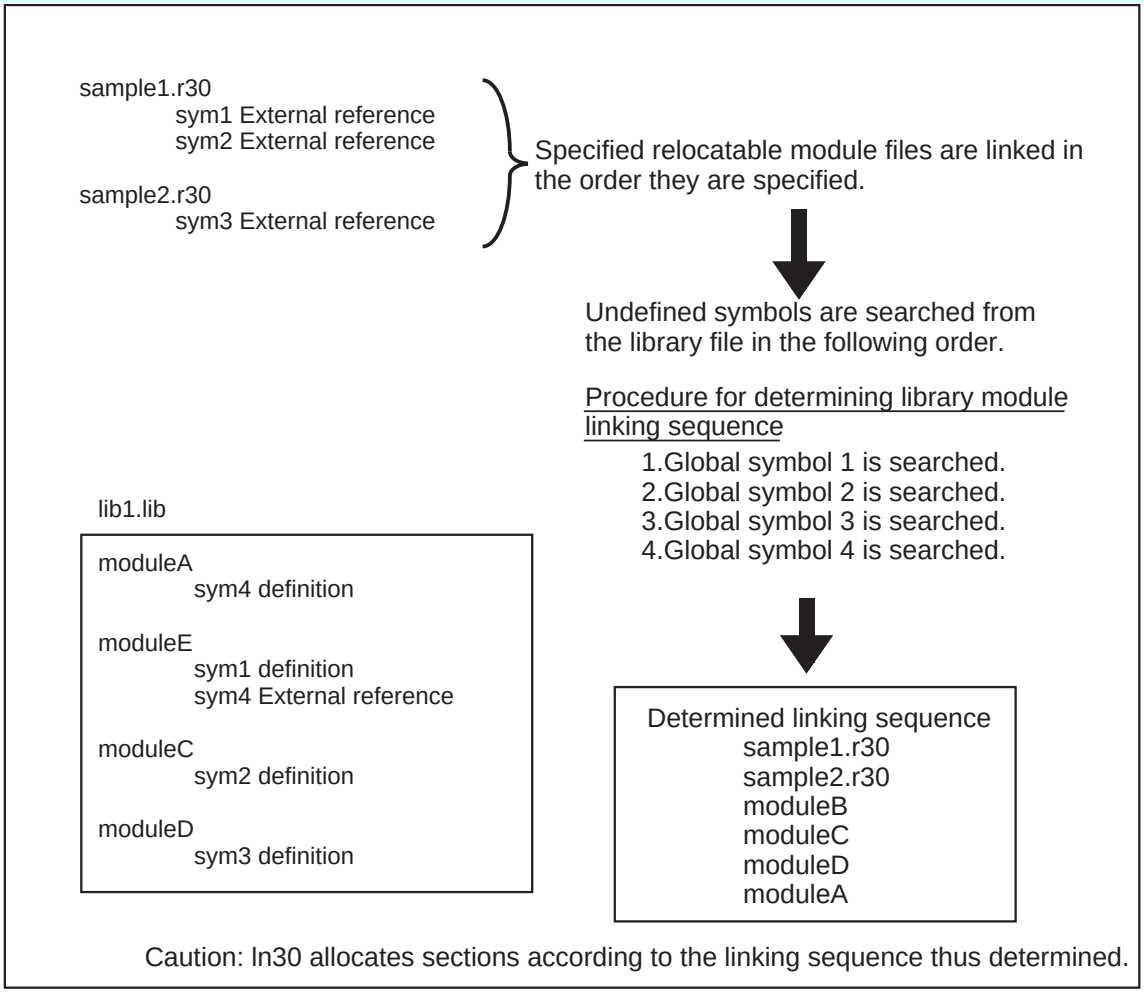
The ln30 editor determines the relocatable modules to be linked in the order described below. A relocatable module that has been determined to be linked is relocated section by section. Sections are relocated in the same way as sections are relocated in a relocatable module file.

1 The ln30 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, ln30 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), ln30 again searches modules in the library file in the order they are entered.

Example of referencing library modules

The following shows an example of how modules in a library file are referenced.
Example: Two relocatable files are linked by entering the command below. In this case, library file "lib1.lib" is referenced as necessary.

```
>ln30 sample1 sample2 -L lib1.lib
```



Management of Include File

The as30 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.

Precautions

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
```

Example of source file (sample.a30)

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

Example of include file (initial.inc)

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

After source file is assembled

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

Address output by as30

Code Selection by AS30

The as30 assembler is designed to choose the shortest code possible from the M16C family's addressing modes. This section outlines the M16C family's addressing modes and explains how to write mnemonics in the source program.

Optimized Code Selection

The as30 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 family 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 Family 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 AS30 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 AS30

The as30 assembler generates optimum-selected or most suitable code for the source statements shown below.

- When jump distance specifier is omitted
  Precautions
  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
  Precautions
  For an addressing mode with displacement, be sure to specify the displacement.
- Combination of the above
  The following explains optimum selection by as30 for each case listed above.

When jump distance specifier is omitted (normally omitted)

The as30 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.

Precautions

- If conditions to perform optimum selection are not met, as30 generates code as directed by directive command ".OPTJ".
- When the branch instruction that is making reference to the global label is optimized, specify “-OGJ(-Oglb_jmp)” of nc30, “-JOPT” of as30 and “-JOPT” options of ln30. However, the directive command “.OPTJ” is ignored where these options were designated.

The following shows instructions selected by as30.

- Unconditional branch instruction
  The shortest instruction possible to branch is selected from jump distances '.A', '.W', '.B', and '.S'.

Precautions

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.

Precautions

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.
The 'ADJNZ' and 'SBJNZ' instruction are equally to the conditional branch instruction optimized.

When instruction format specifier is omitted (normally omitted)

The instruction format specifier normally is omitted.

The as30 assembler performs optimum selection for mnemonics where instruction format specifiers are omitted.

If instruction format specifiers are omitted, as30 first determines the addressing mode before it selects the instruction format.

When addressing mode specifier is omitted

If addressing mode specifiers are omitted, as30 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 following 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.

Precautions
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).

Precautions
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.
- When the following expression that includes a symbol defined by directive command ".SBSYM" is written in the op- code.
  (symbol) - value determined when assembled
  (symbol) + value determined when assembled
  Value determined when assembled + (symbol)

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.

Precautions
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 as30

The examples below show the addressing modes optimum selected by as30 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 AS30 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 family 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 family 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 0FFE00H to 0FFFDBH.
- 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 0FFE00H.

Precaution

For details about the special page vector table, refer to the "M16C Family 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.

Precautions

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      0FFE00H
sub1:         .WORD    label_0 & 0FFFFH ; Special page number 255
sub2:         .WORD    label_1 & 0FFFFH ; Special page number 254
sub3:         .WORD    label_2 & 0FFFFH ; Special page number 253
;
    .ORG      0FFFDAH
sub238:       .WORD    label_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      0F0000H
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 AS30. The following lists the macro functions available for AS30:

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.

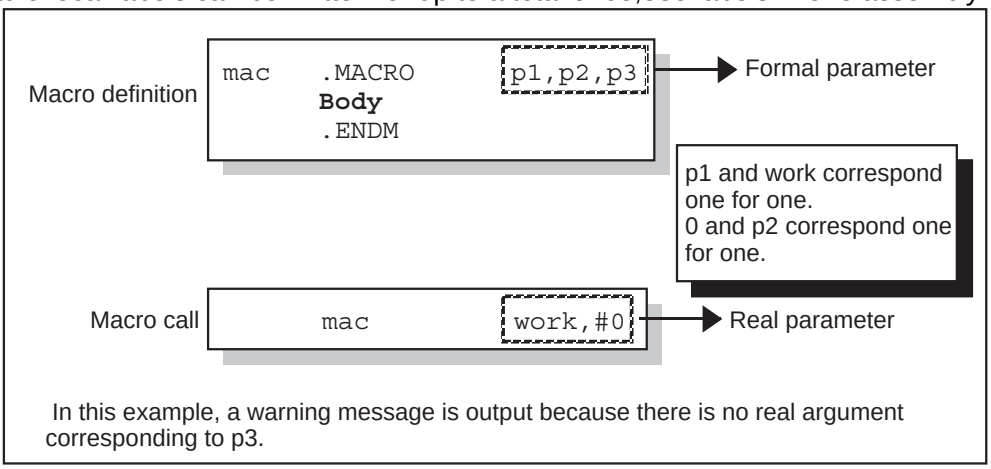
Each macro function is described below.

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.
- 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.

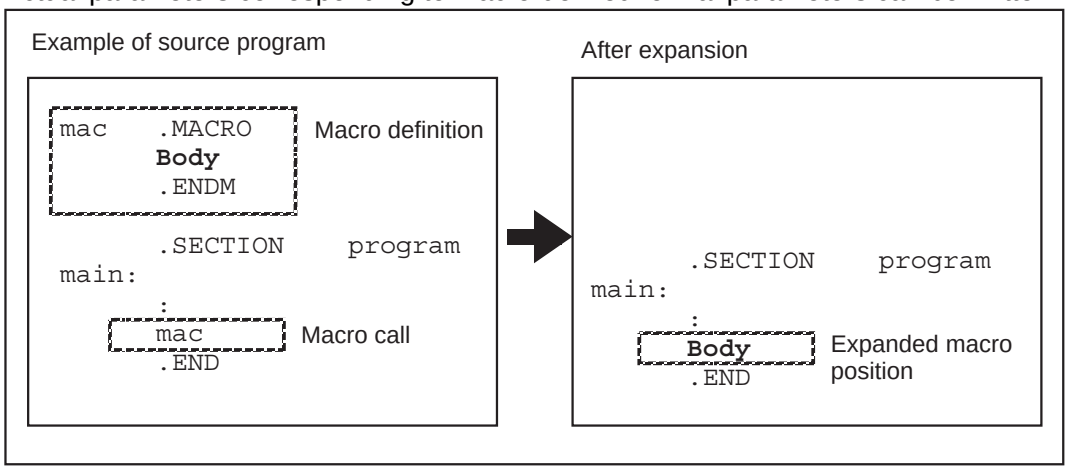


Macro Local Labels

- 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.

Precautions

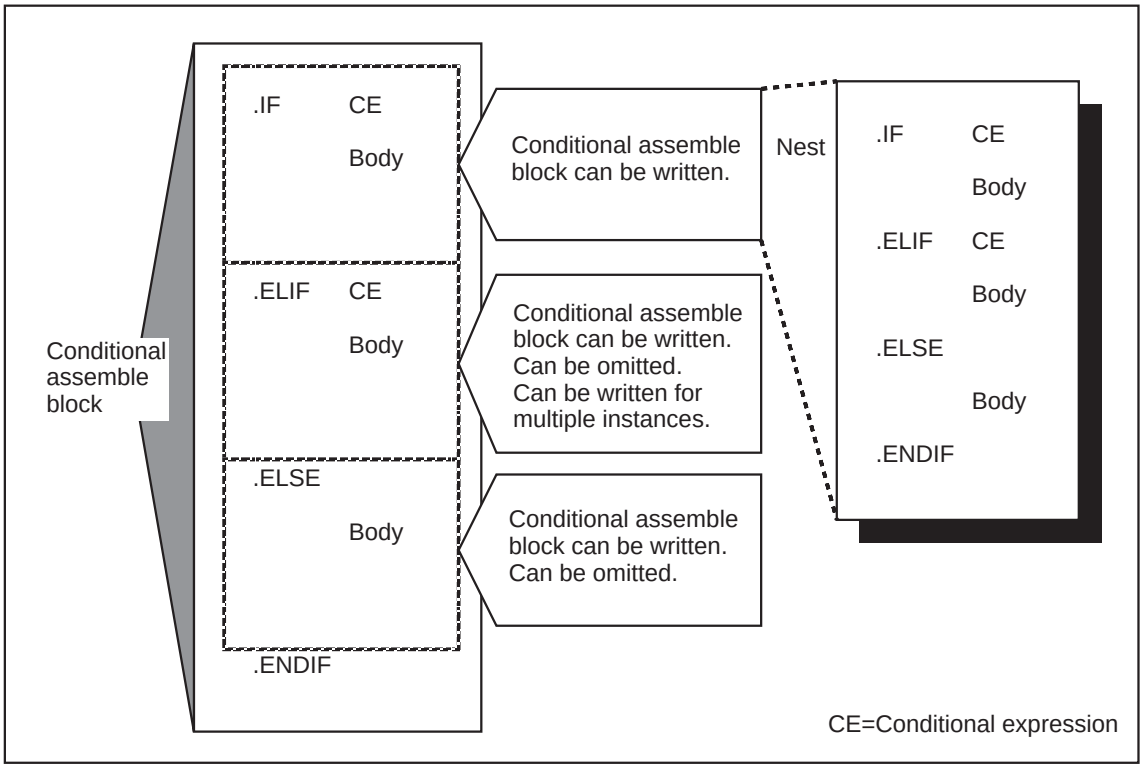
This label is not a macro name. There is no macro call available for repeat macros.

Conditional Assemble Control

The as30 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.

Assebly 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)

```
>as30 sample -Dtype=0
```

Assembled Result 1)

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

Command Input 2)

```
>as30 sample -Dtype=1
```

Assembled Result 2)

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

Command Input 3)

```
>as30 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
```

Structured Description Function

Programming with AS30 allows you to enter structured descriptions using structured description commands.

The following lists the functions of AS30's structured description:

- The assembler generates assembly language branch instructions corresponding to the structured description commands.
- The assembler generates labels indicating the jump address for the generated branch instruction.
- The assembler outputs the assembly languages generated from the structure description commands to an assembler list file. (When a command option is specified)
- A structured description command allows you to select a control block that is made to branch by a structured description statement and its conditional expression. A control block refers to a program section from some structured description statement to the next structured description statement except for assignment statements.

Source Line Information Output

The as30 assembler outputs to a relocatable module file the information that is necessary to implement source debugging of "NC30" and "Macro description of AS30."

Symbol Definition

The as30 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 as30

The as30 assembler references the environment variables listed below.

Environment Variables	Program
AS30COM	as30
BIN30	as30
INC30	as30
LIB30	ln30
TMP30	as30,ln30,lb30

AS30COM

The as30 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 AS30COM

```
SET AS30COM=-L -N -S -T
```

How to clear the settings on AS30COM

```
SET AS30COM=
```

Example for using AS30COM

When environment variable AS30COM is set, as30 sets the command options in the following order.

1 as30 first sets the command options set in AS30COM.
2 as30 sets the command options entered from a command line.

The following shows an example for setting an option to AS30COM, an example for entering a command option from a command line, and an example of a valid command option.

Example for setting up AS30COM

```
SET OPT30=-L -N -S -L
```

Command input example -1

```
as30 -Dsym=0 --N
```

Option that becomes valid when executing as30 -1

```
as30 -Dsym=0 -L -S -T
```

Command input example -2

```
as30 -O\tmp --T -SM -LM
```

Option that becomes valid when executing as30 -2

```
as30 -O\tmp -N -SM -LM
```

BIN30

The assembler driver (as30) invokes the macro processor ,the structured processor and assembler processor residing in the directory you have set.

If this variable is set, always make sure that macroprocessor, structured processor and assembler processor are placed in the directory you have set. Multiple directories can be specified. If multiple directories are specified, as30 searches the directories sequentially from left to right in the order they are written.

INC30

The assembler as30 retrieves include files written in the assembly source file from the directory set in INC30. Multiple directories can be specified. If multiple directories are specified, as30 searches the directories sequentially from left to right in the order they are written.

LIB30

The ln30 retrieves library file to link from directory that is set in thes environment variable. Multiple directories can be specified. If multiple directories are specified, as30 searches the directories sequentially from left to right in the order they are written.

TMP30

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 as30 finishes its processing.

Example of setting environment variable.

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

```
SET INC30=C:\COMMON;C:\PROJECT
```

Output messages

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

Error Messages

This chapter describes the error messages output by each AS30 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.

Precaution

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 AS30 program returns a numeric value to the OS indicating its status at termination.

The table below lists the values that are returned when an error is encountered.

Return value	Content
0	Program terminated normaly.
1	Program was forcibly terminated by input of control C.
2	Error relating to the OS's file system or memory system occured.
3	Error attributable to the file being processed occured.
4	Error in input form the command line occured.

Input/Output Files of AS30

The table below lists the types of files input for AS30 and those output by AS30. Any desired file names can be assigned. However, if the extension of a file name is omitted, AS30 adds the extension shown in ( ) in the table below by default.

as30

Input Files	Output Files
Source file (.a30) Include file (.inc)	Relocatable module file (.r30) Assembler list file (.lst) Assembler error tag file (.atg) Branch Information file (.jin)

ln30

Input Files	Output Files
Relocatable module file (.r30) Library file (.lib) Branch Information file (.jin)	Absolute module file (.x30) Map file (.map) Linkage error tag file (.ltg)

lmc30

Input Files	Output Files
Absolute module file (.x30)	Motorola S format file (.mot) Intel HEX format file (.hex) ID file (.id)

lb30

Input Files	Output Files
Relocatable module file (.r30)	Library file (.lib) Relocatable module file (.r30) Library list file (.lls)

xrf30

Input Files	Output Files
Source file (.a30) Assembler list file (.lst)	Cross reference file (.xrf)

abs30

Input Files	Output Files
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 as30. This file is linked by ln30 to generate an absolute module file.

Format of relocatable module file

The relocatable module file generated by as30 is based on the IEEE-695 format.

Precaution

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 (.a30 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), as30 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 : 0XMSDA
Outputs information on the results of source line processing performed by as30.
Specifically, this information contains the following:

0	X	M	S	D	A	Contents
0-9						Indicates the include file's nest rebel.
	x					Indicates that this line was not assembled in condition assemble.
		M				Indicates that this is a macro and structured statement expansion line.
		D				Indicates that this is a macro definition line.
			S			Indicates that this is a macro expansion line when pre30 does not execute.
				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.

Information in assembler list file

The following information is output to an assembler list file.

- Header information
  The information listed below is output at the beginning of each page of the assembler list file.

```
* R8C/Tiny,M16C FAMILY ASSEMBLER *     SOURCE LIST          Tue Nov 18 12:02:53 1997    PAGE
002

  SEQ.   LOC.     OBJ.                  0XMSDA ....*....SOURCE STATEMENT....7....*....8
```

- Address definition line
  Indicates the line in which location addresses are defined using the directive command ".ORG" .

```
   11                                        .SECTION    RAM,DATA
   12   00400                                    .ORG       000400h
```

- Area definition line
  Indicates the line in which areas are defined using the directive commands ".BLKB", ".BLKW", ".BLKA", ".BLKL", ".BLKF", and ".BLKD".

```
   55                                        .SECTION          ram1,data
   56   00000(000001H)                work1:   .BLKB     1
   57   00001(000001H)                work2:   .BLKB     1
   58   00002(000002H)                work3:   .BLKW     1
   59   00004(000002H)                work4:   .BLKW     1
```

- Comment line
  This line is where only comment are described.

```
   12                                        ;-----------------------------------
   13                                        ; Macro define
```

- Symbol definition line
  Indicates the line in which symbols are defined using the directive command ".EQU".

```
   65   00000001h                     sym1      .EQU      1
   66   00000002h                     sym2      .EQU      2
   67   00000003h                     sym3      .EQU      sym1 + sym2
```

- Bit symbol definition line
  Indicates the line in which bit symbols are defined using the directive command ".BTEQU".

```
   62   1,00000000h                   flag1     .BTEQU    1,0
   63   2,00000000h                   flag2     .BTEQU    2,0
```

- Constant data definition line
  Indicates the line in which data are set in the ROM area using the directive commands ".BYTE", ".WORD", ".ADDR", ".DWORD", ".FLOAT", and ".DOUBLE".

```
  175   0003E   41                  M              .BYTE     "A"
  176   0003F   42                  M              .BYTE     "B"
  177   00040   43                  M              .BYTE     "C"
  178   00041   44                  M              .BYTE     "D"
```

- Macro definition line
  This line is where macros are defined.

```
   46                                        mac5      .MACRO    p1
   47                                    D             .MREPEAT    .LEN{'p1'}
   48                                    D             .BYTE     .SUBSTR{'p1', ..MACREP,1}
   49                                    D             .ENDR
   50                                    D             .ENDM
```

- Label definition line
  This line is where only label name are described.

```
   70   00000                         samp_start:
```

- Mnemonic statement line
  This line is where mnemonics of the M16C Family are described.

```
   71   00000   4100                 S*             BCLR      flag1
   72   00002   4200                 S*             BCLR      flag2
```

- Condition assemble information line
  Indicates the line that was condition assembled. This information is output only when you specified command option (-LI , -LMI or -LMSI).

```
  74                                        .IF       MODE == 1
  75                                 X      MOV.B     #sym1,R0L
  76                                        .ELIF     MODE == 2
  77                                 X      MOV.B     #sym2,R0L
  78                                        .ELSE
  79  00004   B4                       Z     MOV.B     #0,R0L
  80                                        .ENDIF
```

- Macro call line
  This line is where a macro is called. If you specified command option (-LM), this line outputs assembly source lines derived as a result of macro expansion.

```
 173                                        mac5      ABCD
```

- Structured description line
  This line is where program are described in structured directive command.

```
  42                                    for A0 < A1
  43  F800A                          S   ..fr0000:
  44  F800A   C154                   S   CMP.W     A1,A0
  45  F800C   680B                   S   JC        ..fr0002
  46                                        [ WORK_W ] = [ A0 ]
  47  F800E   736F0104               S   MOV.W     [A0],WORK_W
  48                                        [ A0 ] = [ A1 ]
  49  F8012   7376                   S         MOV.W     [A1],[A0]
  50                                        A0 = ++A0
  51  F8014   B2                     S         INC.W     A0
  52                                        A1 = --A1
  53  F8015   FA                     S         DEC.W     A1
  54                                    next
  55  F8016   FEF3                   SB        JMP       ..fr0000
  56  F8018                          S     ..fr0002:
```

- Include file indication line
  This line is where the read-in include file is indicated.

```
  65                                                .INCLUDE    sample.inc
  66                                            1   .SECTION    ram,DATA
  67  00000(000008H)                        1     work8:   .BLKD       1
  68  00008(000004H)                        1     work_4: .BLKF       1
  69                                          1               .SECTION    constdata,ROMDATA
  70  00000   3031323334353637   1       num_val:.BYTE       "0123456789"
                  3839                      1
```

- Assemble result information
  Outputs a total number of errors, total number of warnings, and a total number of list lines derived as a result of assemble processing.

```
Information List

TOTAL ERROR(S)       00000
TOTAL WARNING(S)     00000
TOTAL LINE(S)        00181     LINES
```

- Section information
  Lists the section types, section sizes, and section names.

```
Section List

Attr          Size                    Name
DATA       0000006(00006H)       ram1
CODE       0000066(00042H)       prog1
```

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), as30 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 (asp30): Operand value is not defined
sample.err 72 Error (asp30): 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.

Branch Information File

The as30 generates a branch information file required to optimize the external branch for the a30 file in which the branch instruction referencing the global label exists when “-JOPT” option was specified.

Format of branch information file

The branch information file is only for internal processing of as30 and ln30.
Precaution
Do not edit this file. It should be noted that if it is edited, no subsequent processing is carried out normally.

File name of branch information file

The file name of the branch information file is created by changing the extension of the assembly source file (.a30 by default) to ".jin" (sample.a30 --> sample.jin)

Directory for branch information file generated

The branch information file is generated in the same directory as the relocatable module file.

Absolute Module File

The ln30 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.

Precaution

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), ln30 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, directive command ".ID", ".OFSREG", "PROTECT" and ".VER" 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).

\# VARIABLE VECTOR TABLE INFORMATION
This information includes variable vector table.

\# SPECIAL VECTOR TABLE INFORMATION
This information includes special page vector table.

(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".

Example of Map file

```
##############################################
# (1) LINK INFORMATION                       #
##############################################
ln30 -ms smp

# ID CODE DATA
ID   “Code”

# ROM CODE PROTECT DATA
PROTECT   12H

# LINK FILE INFORMATION
smp (smp.r30)
   Jun 27 14:58:58 1995

##############################################
# (2) SECTION INFORMATION                    #
##############################################
# SECTION                          ATR TYPE      START   LENGTH ALIGN MODULENAME
ram                                  REL DATA      000000  000014               smp
program                              REL CODE      000014  000003               smp
# Total ---------------------------------------------------------------------
                                     DATA                        000014(0000020) Byte(s)
                                     ROMDATA                     000000(0000000) Byte(s)
                                     CODE                        000003(0000003) Byte(s)

##############################################
# (3) GLOBAL LABEL INFORMATION               #
##############################################
work                    000000

##############################################
# (4) GLOBAL EQU SYMBOL INFORMATION          #
##############################################
sym2                    00000000

##############################################
# (5) GLOBAL EQU BIT-SYMBOL INFORMATION      #
##############################################
sym1                    1 000001

##############################################
# (6) LOCAL LABEL INFORMATION                #
##############################################
@ smp ( smp.r30 )
main                    000014   tmp                    00000a

##############################################
# (7) LOCAL EQU SYMBOL INFORMATION           #
##############################################
@ smp ( smp.r30 )
sym3                    00000003

##############################################
# (8) LOCAL EQU BIT-SYMBOL INFORMATION       #
##############################################
@ smp ( smp.r30 )
sym4                    1 000000
```

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), ln30 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 (ln30): smp2.r30 : Absolute-section is written after the absolute-section 'ppp'
smp.inc 2 Error (ln30): 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 lmc30 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), lmc30 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.

Precaution
IF the address value exceeds 1Mbytes of machine language file, the file of Original HEX format for 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.

ID File

When you specify command option (-ID) or the assembler directive command (.ID), lmc30 outputs ID code to a file.

Format of ID file

This file is output in a text format that can be output to a screen and printer. By referencing thisfile, confirm ID code of ID code function.

The ID file is output in order command option information, ID code store address.

```
-IDCodeNo1
FFFFDF   :43
FFFFE3   :6F
FFFFEB   :64
FFFFEF   :65
FFFFF3   :4E
FFFFF7   :6F
FFFFFB   :31
```

File name of ID file

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

Directory for ID file generated

The files are generated in the current directory.

Library File

The lb30 librarian generates one library file from the relocatable module files generated by as30 by integrating them as modules into a single file.

Format of library file

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

Precaution

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 lb30 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.

Precaution
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 (lb30) for M16C Family 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 xrf30 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 abs30 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 AS30.

Precautions on Entering Commands

- Use the MS-DOS prompt to input a command.
- AS30 allows use of the period in only one place of a file name.

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.

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.
- Command option
  Specify command options on the command line to use the functions of AS30 programs.

Rules for Entering Command Line

When starting up each AS30 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 2048 characters (bytes).

Precaution

The number of characters may be limited below the above specification depending on the operating environment (type of OS) of AS30.

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 512 characters (bytes) including directory specification. 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 OS in addition to the above rules. Refer to the user's manual of your OS for details.

Precaution

AS30 allows use of the period in only one place of a file name. Furthermore, some AS30 programs restrict file name extensions (characters following the period) also. Refer to the method for starting up each AS30 program for details.

Command Options

- Always be sure to add a hyphen (-) when entering a command option.

Method for Operating as30

This section explains the method for operating as30 to utilize its functions. The basic function of as30 is to generate a relocatable module file from the assembly source file.

Command Parameters

The table below lists the command parameters of as30.

Parameter name	Function
Source file name	Source file name to be processed by as30.
-.	Disables message output to a display screen.
-A	Evaluates mnemonic operand.
-C	Indicates contents of command lines when as30 has invoked mac30, pre30 and asp30.
-D	Sets constants to symbols.
-finfo	Generates inspector information.
-F	Fixes the file name of ..FILE expansion to the source file name.
-H	Header information is not output to an assembler list file.
-I	Specify an include file search directory.
-JOPT	Optimizes the branch instruction referencing the global label.
-L	Generates an assembler list file.
-M	Generates structured description command variables in byte type.
-M60	Generates code that conforms to the M16C/60 group.
-M61	Generates code that conforms to the M16C/61 group.
-N	Disables output of macro command line information.
-O	Specifies a directory to which the generated file is output.
-P	Processes structured description command.
-PATCH(6N)_TA -PATCH(6N)_TAn	Generates code to escape precautions on the timer functions for three-phase motor control
-R8C	Generates code that conforms to the R8C Family. (Address area is 0H to 0FFFFH.)
-R8CE	Generates code that conforms to the R8C Family. (Address area is 0H to 0FFFFFH.)
-R8Cxx	Generates a code that avoids notes of Clock Synchronous Serial I/O with Chip Select (SSU) or I^2C bus Interface (IIC).
-S	Specifies that local symbol 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 as30.

Order in which to specify command parameter

- Command parameters can be specified in any desired order.
  as30 (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.

Precaution
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.

Precaution
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.

Precaution
Command options can only be nullified in as30. Therefore, this function cannot be used when starting up any other program.

Example:

- Option L only is valid.

```
>as30 sample -L
```

- Option S only is valid.

```
>as30 sample -S
```

- Option S only are valid.

```
>as30 sample -L -S --L
```

- Options L only are valid.

```
>as30 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 "INC30" is set, the software also searches the directory that is set in INC30.

as30 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 as30 is processing.
- This command option disables unnecessary messages such as copyright notes from being output to the screen when executing as30 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

```
>as30 -. sample
```

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

```
>as30 -. sample

sample.a30 2 Error (as30) : Section type is not appropriate
```

-A

Evaluate Mnemonic Operand

Function

- The software outputs a warning if for a mnemonic where both immediate and address values can be written, the symbol '#' to indicate that the operand is an immediate is not written in it.

Precautions
The warning is output if the operand is a numeric value except labels or a symbol whose value is fixed when assembled.

Description rule

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

Description example

```
>as30 -A sample
```

-C

Indicates Command Invocation Line

Function

- In cases when a command option is specified in environment variable (AS30COM), if this option is specified you can confirm the command options set when invoking macroprocessor, structured processor and assembler processor from as30 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 AS30COM, the following output will be obtained.

```
>as30 -C -N sample
```

- This information is displayed beginning with the next line following "All Rights Reserved." that is output when AS30 starts up normally.

```
>as30 -C -N sample

( sample.a30 )
mac30 -L -T sample.a30
macro processing now

asp30 -L -T sample.a30
assembler processing now
TOTAL ERROR(S)          00000
          :
```

```
>as30 -. -C -N sample
```

- If this command option is combined with an option to disable message output to a screen, the following output will be obtained.

```
>as30 -. -C sample
mac30 -L -T sample.a30

asp30 -L -T sample.m30
```

-D

Sets Symbol Constant

Function

- The software sets values to symbols.
- The value is handled as an absolute value.

Precaution

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.

```
>as30 -Dname=1 sample
```

- This example sets 1 to symbols name and symbol.

```
>as30 -Dname=1:symbol=1 sample
```

- This example defines a symbol named name for files sample1 and sample2.

```
>as30 -Dname=1 sample1 sample2
```

-finfo

Generates inspector information

Function

- Outputs either each item of information generated by the '-finfo' option in NC30 or inspector information described in assembler directives to a relocatable module file.

Note

With a TM in use, this option is chosen by default.

Description rules

- You can put this option anywhere in a command line.
- Use lowercase letters only, since uppercase letters and lowercase letters are discriminated.

Description example

```
> as30 -L -S -finfo sample
```

-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

```
>as30 -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.a30 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.

Precautions

When generating an assembler list file to be processed by abs30, 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.

```
>as30 -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.

```
>as30 -I\work\include
```

-JOPT

Optimizes the branch instruction referencing the global label

Function

- This option is used to optimize the branch instructions (JMP & JSR) that are making reference to the global label.

Precautions

When this option is specified, always specify the nc30 “-OGJ(-Oglb_jmp)” and ln30 “-JOPT” options.

When this option was specified, the branch information file (extent: .jin) is generated.

Do not edit the branch information file. Also, avoid using the extent “.jin”.

When the directive command “.OPTJ” is being used simultaneously with this option, this option becomes effective.

Description rule

- This option can be written at any desired position in a command line.

Description example

```
>as30 -JOPT sample
```

-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 "AS30COM".

Format specifier	Function
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 expansion sections are output to the assembler list file.

Description example

```
>as30 -LIM sample
>as30 -CDLSMI 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
```

-M60/-M61

Control code generation

Function

- The software processes the following description:

Option	content
-M60	The mnemonic ‘NOP’ is added after the line witch is written mnemonics 'SHL','SHA' and 'ROT'. The mnemonic ‘JMP.B’ is added before the line witch is written the mnemonic 'JMP.A' and ‘JSR.A'.
-M61	The as30 does not process. Refer to the ln30's processing.

Precaution
“-R8C” option cannot be specified at the same time as this option.

Description rules

- This option can be written at any desired position in a command line.

Description example

```
>as30 -M61 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.

Precaution
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 "AS30COM". Refer to "Example for using AS30COM" for details on how to set.

Description example

```
as30 -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.

```
>as30 -Oc:\work\asmout sample
```

- The relocatable module file is generated in the tmp directory that is the parent directory of the current directory.

```
>as30 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.

```
>as30 -Oc:\work\asmout sample -L -T
```

-P

Process Structured Description Command

Function

- The software processes the structured description commands written in the assembly source file.

Description rule

- This option can be specified at any position of the command line.
- When using structured description commands, be sure to specify this option.

Description example

```
>as30 -P -LS sample
```

The software processes the structured description commands written in the assembly source file and outputs the expanded sections to an assembly list file.

-PATCH(6N)_TA/-PATCH(6N)_TAn

Escaping precautions on the timer functions for three-phase motor control

Function

- Generates code to escape precautions on the timer functions for three-phase motor control.

Precaution

Refer to “TECHNICAL NEWS” for details about the precautions discussed here.

- The escape code is generated only when a value is written to the address indicated by the Timer A1-1 Register (TA11), Timer A2-1 Register (TA21) or Timer A4-1 Register (TA41) by using the MOV instruction (word length). (The above address refers only to one that is fixed when assembled.)

Option specifier	Object address
-PATCH_TA, -PATCH_TAn	TA11 is 302H address TA21 is 304H address TA41 is 306H address
-PATCH6N_TA, -PATCH6N_TAn	TA11 is 1C2H address TA21 is 1C4H address TA41 is 1C6H address

Precaution

“-R8C” option cannot be specified at the same time as this option.

Description rule

- This command option can be specified at any position in the command line.
- Any decimal number from 0 to 99 can be specified for “n” in “–PATCH_TAn”.
- This option must always be specified in uppercase letters.

Description example1

source file description example)

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

-PATCH_TA specification, the list file output example)

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

The same MOV mnemonic written here is generated as escape code.

Description example2

source file description example)

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

-PATCH_TA2 specification, the list file output example)

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

Two or more of the NOP mnemonic specified by “n” and the same MOV mnemonic written here are generated as escape code.

-R8C/-R8CE

Control code generation

Function

- Generates a code that conforms to the R8C Family series.

Option	Address area
-R8C	0H to 0FFFFH
-R8CE	0H to 0FFFFFH

Precaution
Symbol setting option “-D__R8C__=1" is added.
“-M60”, “-M61”, ”-PATCH(6N)_TA” and “-PATCH(6N)_TAn” option cannot be specified at the same time as this option.

Description rules

- This option can be written at any desired position in a command line.

Description example

```
>as30 –R8C sample
```

-R8Cxx

Control code generation

Function

- Generates a code that avoids notes of Clock Synchronous Serial I/O with Chip Select (SSU) or I^2C bus Interface (IIC). (The above register's address refers only to one that is fixed when assembled.)
- When this option is specified, "r8ctiny.txt" in environment variable "LIB30" directory is referred to.
- When the group name applicable to SSU or IIC function is specified, a message "R8C/xx group in information file 'r8ctiny.txt' is used." is outputted.
- When this option is specified, "-R8C" option is added.

Precaution

Refer to RENESAS TECHNICAL UPDATE about the notes.

In case of High-performance Embedded Workshop, specify this option in "Options" -> "Renesas M16C Standard Toolchain" -> "CPU" menu.

In case of TM, specify this option in "Option Browser" -> "CFLAGS" -> "General" or "AFLAGS" -> "Select target." menu.

"-M60", "-M61", "-PATCH(6N)_TA" and "-PATCH(6N)_TAn" option cannot be specified at the same time as this option.

Description rules

- This option can be written at any desired position in a command line.
- This option must always be specified in uppercase letters.
- Specify this option like –R8C(group name).

Precaution

This option is not necessary in case of a group that does not have SSU or IIC function.

Description example

```
>as30 –R8C14 sample
```

Generates a code that avoids notes of SSU function of R8C/14.

Example of source file)

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

Example of list file)

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

-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.

Precaution

The map file (.map) output by ln30 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 "AS30COM". Refer to "Example for using AS30COM" for details on how to set.

Description example

- Local symbol information in sample.a30 is output to sample.r30.

```
>as30 -S sample
```

- Local symbol information and system label information in sample.a30 is output to sample.r30.

```
>as30 -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 "AS30COM". Refer to "Example for using AS30COM" for details on how to set.

Description example

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

```
>as30 -T sample
```

-V

Indicates Version Number

Function

- When this option is specified, the software indicates the version numbers of all programs included with AS30 and terminates processing.

Precaution

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

```
>as30 -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.

```
>as30 -Xedit sample
```

Error Messages of as30

'#' 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.
'.EINSF' is missing for '.INSF'
? .EINSF, used with .INSF in a pair, is missing.
! Check where .INSF is put.
'.ID' is duplicated
? .ID is specified more than once in the file.
! .ID can be written only once in a file. Delete extra .ID's.
'.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.
'.INSF' is missing for '.EINSF'
? .INSF, used with .EINSF in a pair, is missing.
! Check where .EINSF is put.
'.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.
'.PROTECT' or '.OFSREG' is duplicated
? .PROTECT or .OFSREG is specified more than once in the file.
! .PROTECT and .OFSREG can be written only once in a file. Delete extra .PROTECT's or .OFSREG's.
'.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 'TMP30' 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.

Directive command '.RVECTOR' can't be described
? The directive command '.RVECTOR' cannot be written here.
! If a variable vector table is to be automatically generated, do not write a program in the 'vector' section.

Directive command '.SVECTOR' can't be described
? The directive command '.SVECTOR' cannot be written here.
! If a special page vector table is to be automatically generated, do not write a program in the 'svector' section.

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.

Function information is not defined
? Function information, which is inspector information, has not been defined.
! Define the function information as required.

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.

Initialization function definition of the section is not appropriate
? The section initialization function that involves use of C language startup is defined incorrectly.
! Check how the section initialization function is defined.

Interrupt number was already defined
? The software interrupt number was already defined.
! Change the software interrupt number.

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

Invalid directive command which isn't supported in '-R8C'
? A directive command that cannot be specified simultaneously with the '-R8C' option is written.
! Check the content of the directive command written.

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

Invalid mnemonic which isn't supported in '-R8C'
? The mnemonic which isn't supported in R8C Family is described.
! Specify the mnemonic correctly again.

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 xxx
? The location counter exceeded xxx.
! 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 name is not appropriate
? Section name is not appropriate.
! Change the section name.

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.

Special page number was already defined
? Special page number was already defined.
! Change the special page number.

Specifies option that can't use with 'xx'
? The option which can not be specified simultaneously with 'xx' is specified.
! Specify the command option correctly again.

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 as30

'-JOPT' and '.OPTJ' are specified
? '-JOPT' option and the directive command '.OPTJ' are specified.
! The directive command '.OPTJ' is ignored.
'.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.
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.

Location counter exceed xxx
? The location counter exceeded xxx.
! Check the operand value of .ORG. Rewrite the source 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 ln30

This section describes how to use the functions of ln30. The basic function of ln30 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 ln30.

Parameter name	Function
File name	Relocatable module filename to be processed by ln30
-.	Disable message output to screen.
-E	Specifies start address of absolute module.
-G	Outputs source debug information to absolute module file.
-JOPT	Optimizes the branch instruction referencing the global label.
-L	Specifies library file to be referenced.
-LOC	Specifies section allocation sequence.
-LD	Specifies directory of library to be referenced.
-M	Generates map file.
-M60	Generates code that conforms to the M16C/60 group.
-M61	Generates code that conforms to the M16C/61 group.
-MS	Generates map file that includes symbol information.
-MSL	Generates map file that includes full name of symbol more than 16 characters.
-NOSTOP	Outputs all encounters errors to screen.
-O	Specifies absolute module file name.
-ORDER	Specifies section address and allocation sequence.
-R8C	Generates code that conforms to the R8C Family (Address is 0H to 0FFFFH).
-R8CE	Generates code that conforms to the R8C Family (Address is 0H to 0FFFFFH).
-T	Generates link error tag file.
-U	Outputs a warning for the unused function names.
-V	Indicates version number of linkage editor.
-VECT	Sets the value for the free area in generating the variable vector table.
-VECTN	Sets the address of a software interrupt number.
-W	On the occurrence of a warning, no absolute module file is generated.
@	Specifies command file.

Rules for Specifying Command Parameters

Follow the rules described below when you specify command parameters for ln30.

Order in which to specify command parameters

- Relocatable module file names and command options can be specified in any desired order.
  >ln30 (command options) (relocatable module file)
  >ln30 (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 ln30 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 (LIB30). If the relevant file cannot be found, ln30 searches the current directory. Or if a directory is specified by command option (-LD), ln30 searches it and if no relevant file is found in this directory, ln30 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 ln30 editor determines absolute addresses section by section as it generates an absolute module file.
- When invoking ln30, 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 ln30 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:

```
ln30 @cmdfile
```

- A directory path can be specified in the command file name.
- If no file exists in the specified directory path, ln30 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 ln30.

- 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 2047 characters. If this limit is exceeded, ln30 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
```

Command Options of ln30

This section explain how to specify command options when using ln30.

-.

Disables Message Output to Screen

Function

- The software does not output messages when ln30 is processing.
- Error messages are output to screen.

Description rule

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

Description example

```
ln30 -. 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".

```
ln30 sample1 sample2 -E num
```

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

```
ln30 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.

Precaution

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 as30, it cannot be debugged at the source line level even when you have specified this option (-G) when executing ln30.

- 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

```
ln30 -G sample1 sample2
```

-JOPT

Optimizes the branch instruction referencing the global label

Function

- This option is used to optimize the branch instructions (JMP & JSR) that are making reference to the global label.

Precautions

- When this option is specified, always specify the nc30 “-OGJ(-Oglb_jmp)” and as30 “-JOPT” options.
- When this option was specified, the branch information file (extent: .jin) is generated. Do not edit the branch information file. Also, avoid using the extent “.jin”.
- When the directive command “.OPTJ” is being used simultaneously with this option, this option becomes effective.
- When this option is specified, execute nc30, as30 and ln30 at the same directory.

Description rule

- This option can be written at any desired position in a command line.

Description example

```
>ln30 -JOPT sample
```

-L

Specifies Library File Name

Function

- Specify the library file name to be referenced when linking files.
- The ln30 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

```
>ln30 sample1 sample2 -L lib1
```

The "lib1.lib" file in the current directory or the directory specified in environment variable (LIB30) is referenced as necessary.

```
>ln30 sample1 sample2 -L work\lib1
```

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

```
>ln30 sample1 sample2 -L lib1,lib2
```

The "lib1.lib" and "lib2.lib" files in the current directory or the directory specified in environment variable (LIB30) 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 ln30 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.

```
>ln30 sample1 sample2 -LD \work\lib -L lib1
```

- The \work\lib\lib1 and \work\tmp\lib2 files are referenced.

```
>ln30 sample1 sample2 -LD \work\lib -L lib1 -LD \work\tmp -L lib2
```

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

```
>ln30 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".
- This option is for applications that run a program on the RAM.

Precaution

1. It is possible that the program relocated by specification of ALIGN will not operate normally. Therefor, when using the "-LOC" option, relocate a section whose beginning address is even to an even address, and one whose beginning address is odd to an odd address.
2. The only function of the -LOC option is to register the defined program to the specified address. It does not send the program to the address area during execution of the application.

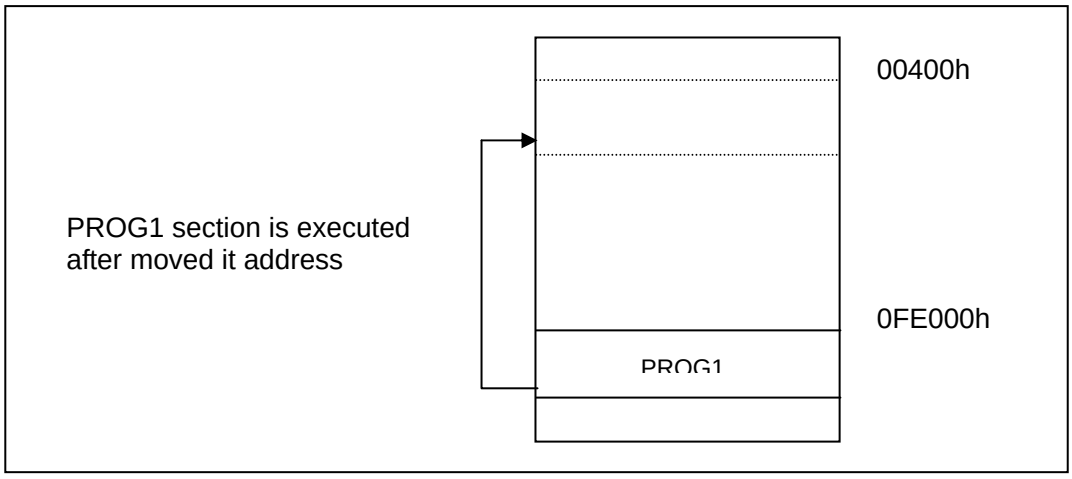
Description rule

- This command option can be specified at any position in the command line.
- 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 shows a case where as depicted in the diagram below, the section name PROG1 stored in the ROM at the address FE000h is transferred to the RAM at the address 00400h before being executed in the RAM (address 00400h).
  In this case, specification for ln30 is written as shown below.

```
>ln30 -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.

```
>ln30 -M sample1 sample2
```

-M60/-M61

Control code generation

Function

- The spftware processes the following description:

Option	content
-M60	If 'JMP.S' instruction exist at end of bank when linking, warning message is output. In this case, cope with the warning by directive command '.SJMP'.
-M61	If 'JMP.S' instruction exist at end of bank when linking, warning message is output. In this case, cope with the warning by directive command '.SJMP'.

Precaution
"-R8C" option cannot be specified at the same time as this option.

Description rules

- This option can be written at any desired position in a command line.

Description example

```
>ln30 -M60 sample
```

-MS/-MSL

Outputs Symbol Information to Map File

Function

- The fullname of symbolmore 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

```
>ln30 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

```
>ln30 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 ln30.
- 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.

```
>ln30 sample1 sample2 -O abssmp
```

- A "abssmp.x30" file is generated in the "\work\absfile" directory.

```
>ln30 -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.

Precaution

If the start address is specified for an absolute section, ln30 outputs an error.

- If you do not specify the start address, ln30 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.
- When the name of the section not existing is specified, the section name is disregarded.

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

```
>ln30 sample1 sample2 -ORDER main,sub,dat                (1)
>ln30 sample1 sample2 -ORDER main=0f0000,sub,dat         (2)
```

(1) Sections are allocated in order of main, sub, and dat beginning from address 0H.
(2) Sections are allocated in order of main, sub, and dat beginning from address 0f000H.

If the name of the section not existing “noprog” is specified:

```
>ln30 sample1 sample2 -ORDER main=0f000,noprog,dat                (3)
>ln30 sample1 sample2 -ORDER main=0f000,noprog=0f2000,dat         (4)
>ln30 sample1 sample2 -ORDER main=0f000,noprog=0f2000,dat=0f3000  (5)
```

(3) Sections are allocated in order of main, dat beginning from address 0f000H.
(4) “main” section is allocated from address 0f000H. “dat” section is allocated from address 0f2000H.
(5) “main” section is allocated from address 0f000H. “dat” section is allocated from address 0f3000H.

-R8C/-R8CE

Control code generation

Function

- Generates a code that conforms to the R8C Family.

Option	Address area
-R8C	0H to 0FFFFH
-R8CE	0H to 0FFFFFH

Precaution
"-M60" and "-M61" option cannot be specified at the same time as this option.
The value set to the option function select register is outputted by message
"The Value of option function select register is xxH".

Description rules

- This option can be written at any desired position in a command line.

Description example

```
>ln30 –R8C sample
```

-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.

```
>ln30 sample1 sample2 -T
```

-U

Outputs a warning for the unused function names.

Function

- Outputs a warning for the unused function names that are written in the C language source file.

Precaution

1. Before deleting the unused functions, check to see that they really are unnecessary functions.
2. This option is effective when the “-finfo” option of NC30 and AS30 is specified.
3. When calling any assembler function from a C language program, always be sure to write the assembler directive “.INSF” and “EINSF” for the assembler function.
4. If any interrupt function is written with an assembler function, it will be alerted as an unused function by outputting a warning. (This applies when the assembler directive “.INSF”and “.EINSF” is written.)
5. The names listed below are outside the scope of search for unused functions.
   Assembler function name : start
   C language function name : main, Run-time and standard library functions

Description rule

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

Description example

```
>ln30 –U sample
```

-V

Indicates Version Number

Function

- The software indicates the version number of ln30.

Precaution

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

```
>ln30 -V
```

-VECT

Sets a value for the free area in generating the variable vector table

Function

- Set a value for the free area left after a variable vector table was automatically generated (software interrupts not set by the assembler directive ".RVECTOR"). A value or a global label name can be specified for the value to be set for the free area. Note that the set value is handled as 4-byte data.

Precautions

1. When this option is specified, a variable vector table is automatically generated. For details about automatic setting of a variable vector table, refer to the assembler directive ".RVECTOR."
2. If the "-VECTN" option is specified, the "-VECTN" option has priority.

Description rule

- Input this option using a form like -VECT (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.
- Specify this option only and nothing else.

Description example

```
>ln30 -VECT stop sample1 sample2     ...(1)
>ln30 sample1 sample2                ...(2)
```

(1) Sets the address of "stop" for the free area in generating the variable vector table.
(2) Sets the address of "__dummy_int" for the free area in generating the variable vector table.

-VECTN

Set the address value of a specified software interrupt number.

Function

- Set the address value of a specified software interrupt number.

Precautions

- When this option is specified, a variable vector table is automatically generated. For details about automatic setting of a variable vector table, refer to the assembler directive ".RVECTOR."
- If the software interrupt number specified in the assembler directive ".RVECTOR" is specified in this option, an error is assumed.
- This option has priority over the "-VECT" option.

Description rule

- Make sure this option is entered in the form like *–VECTN (value or symbol name), software interrupt number*.
- To specify multiple software interrupt numbers, specify them in the form like *(value or symbol name), software interrupt number, (value or symbol name), software interrupt number*.
- Always be sure to enter a space between this option and the value or label name.
- Always be to specify values in hexadecimal.
- If the value begins with an English alphabet (a, b, c, d, e, f), be sure to write 0 at the beginning.
- Be sure to separate a value and a software interrupt number with a comma.
- No spaces or tabs can be entered before and after a comma.
- Always make sure software interrupt numbers are entered in decimal.
- This option can be written at any position in the command line.

Description example

```
>ln30 -VECTN stop,20 sample1 sample2     ...(1)
>ln30 -VECTN stop0,20,stop1,21 sample1 sample2       ...(2)
```

(1) The address value of "stop" is set for software interrupt number = 20.
(2) The address values of "stop0" and "stop1" are set for software interrupt numbers = 20 and 21, respectively.

-W

On the occurrence of a warning, no absolute module file is generated.

Function

- On the occurrence of a warning, no absolute module file is generated.
- On the occurrence of a warning, “10” is returned to the return value of OS.

Precaution

When this option is not specified, “0” is returned to the OS return value.

Description rule

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

Description example

```
>ln30 -W sample1 sample2
```

@

Specifies Command File

Function

- The software starts up ln30 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

```
>ln30 @cmdfile
```

Error Messages of ln30

'-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.
'-VECT' option parameter 'symbol' is undefined
? The symbol 'symbol' specified with -VECT is not defined yet.
! Check the symbol name.
'-VECTN' option parameter 'symbol' is undefined
? The symbol ‘symbol’ specified in -VECTN is undefined.
! Check the symbol name.
'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 generate automatically the variable interrupt vector table
? The linker cannot automatically generate a variable vector table.
! If a variable vector table is to be automatically generated, do not write a program in the "vector" section.

Can't generate automatically the special page vector table
? The linker cannot automatically generate the special page vector table.
! If the special page vector table is to be automatically generated, do not write a program in the "svector" section.

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.

Command-file line characters exceed
? The number of characters per line in the command file exceeds the limit.
! 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.

Error occurred in executing 'xxx'
? Error occurred in executing 'xxx'.
! Check the error message of 'xxx'.

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 as30.

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 as30.

Illegal format 'file' :it's not jin file
? The format of the file 'file' is illegal. That is not a branch information file (.jin).
! Check to see that the branch information file is one that was created by as30.

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 lb30.

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 as30.

Interrupt number 'X' is multiple defined
? The software interrupt number 'X' is multiple defined.
! Check the software interrupt number.

Interrupt number 'X' is multiple defined by '-VECTN' and '.rvector'
? The software interrupt number 'X' is multiple defined by '-VECTN' and '.rvector'.
! Check the software interrupt number.

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 as30.

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

Special page number 'X' is multiple defined
? Special page number 'X' is multiple defined.
! Check the special page number.

Special page number 'X' is multiple defined
? Special page number 'X' is multiple defined.
! Check the special page number.

Specified an option that can't be used with 'option'
? The option that cannot be used with 'option' at the same time is specified.
! Check the option.

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.

Wrong value is specified by option "-loc".
? There is a wrong in the address, which was specified by "-loc"
! Confirm the precaution of "-loc".

Zero division exists in the expression
? Expression for relocation data calculations contain a divide by 0 operation.
! Rewrite the expression correctly.

Warning Messages of ln30

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

'section' data exceed xxx
? The section data exceeds address xxx.
! 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 as30.

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.
Directive command '.ID' is duplicated
? .ID is specified more than once in the file.
! .ID can be written only once in a file. Delete extra .ID's.
Directive command '.PROTECT' or '.OFSREG' is duplicated
? .PROTECT or .OFSREG is specified more than once in the file.
! .PROTECT and .OFSREG can be written only once in a file. Delete extra .PROTECT's or .OFSREG's.
Global function ‘xxx’ is never used
? The global function ‘xxx’ is not used once.
! Check to see if it is a necessary function.
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.
Local function ‘xxx’ is never used
? The local function ‘xxx’ is not used once.
! Check to see if it is a necessary function.
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 AS30 program. Regenerate the file as necessary. If this error occurs, please contact tool support personnel.
Section type mismatch 'section'
? Sections of the same name 'section' have different section types.
! Check the section types in the source file
The free area’s address in vector table isn’t specified.
? The free area’s address in vector table isn’t specified.
! Check the free area in vector table.

Method for Operating lmc30

This section explains how to operate lmc30. The primary function of lmc30 is to generate a machine language file in the Motorola S format from the absolute module files generated by ln30.

Command Parameters

The table below lists the command parameters available for lmc30.

Parameter name	Function
File name	Absolute mdoule file name to be processed by lmc30.
-.	Disables message output to screen.
-A	Specifies an address range of output data.
-E	Sets the starting address.
-F	Sets data in a space area.
-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.
-ofsregx	Set the option function select register
-protect1	Set levet1 for ROM code protect function
-protectx	Set the ROM code protect control address
-R8C	Generates code that conforms to the R8C Family (Address is 0H to 0FFFFH).
-R8CE	Generates code that conforms to the R8C Family (Address is 0H to 0FFFFFH).

Rules for Specifying Command Parameters

Follow the rules described below when you specify the command parameters for lmc30.

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)

```
>lmc30 (command option) (absolute module file name)
```

Absolute module file name (essential)

- Specify the absolute module file generated by ln30.
- 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.

lmc30 Command Options

This section explains how to specify the command options of lmc30.

-.

Disables Message Output to Screen

Function

- The software does not output messages when lmc30 is processing.
- Error messages are output to screen.

Description rule

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

Description example

```
>lmc30 -. debug
```

-A

Specifies an address range of output data

Function

- Specifies an address range for machine-language data to be output to a file generated.
- You can use one of two ways given below to choose this option.
  1. You specify the starting address and the ending address of the output.
  2. You specify the starting address of the output only.

Description rules

- Specify this option either in the form of "-A (starting address: ending address)" or "-A (starting address)".
- Put at least one space between this option and the starting address.
- Be sure to give an address in hexadecimal.
- Specify this option ahead of specifying a file name.
- With the starting address of the output only specified, the maximum address of the data registered in the absolute module file becomes the ending address.
- If the starting address exceeds the ending address, an error occurs.
- As for a specified address range in which no data are present, the specified data will be output if the option -F, which is an option for setting data in a space area, is chosen, or nothing will be output if this option is not chosen.
- An error results if the start address value is greater than the maximum address for data that is registered in the absolute module file.
- An error results if the end address value is smaller than the minimum address for data that is registered in the absolute module file.
- An error results if the start address and the end address values are the same.

Description examples

```
>lmc30 -A 1000:11FF sample
```

The starting address of a specified address range is set to 1000H, and the ending address to 11FFH.

```
>lmc30 -A 1000 sample
```

In the specified range of addresses, 1000H is the start address value, and the maximum address for data that is registered in the absolute module file *sample* is the end address value.

-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.

Precaution

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

Description example

```
>lmc30 -E 0f0000 debug
```

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

```
>lmc30 -E 8000 debug
```

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

-F

Sets data in a space area

Function

- Outputs arbitrary data to addresses holding no data within a specified absolute module file.
- Following three specifications are possible for this option:
  1. Specification of only the data value that is to be set in a free area
  2. Specification of the data value that is to be set in a free area and the start address value of that area
  3. Specification of the data value that is to be set in a free area and the start address and end address values of that area

Description rules

- This option must be specified in the form like "–F (free-area set data value)", "–F (free-area set data value: start address value)" or "–F (free-area set data value: start address value: end address value)."
- Put at least one space between this option and the data.
- Be sure to give an address in hexadecimal.
- Specify this option before of specifying a file name.
- An error results if the start address value is greater than the end address value.
- When using this option in combination with the "–A" option, an error will result unless the output range of the free-area set data is within the address range specified by the "–A" option.
- It is only when the start address value is greater than the maximum address for data that is registered in the absolute module file and the start address and end address values both are specified that the free-area set data is additionally output to the specified address range. An error results if the start address value only is specified.
- If the end address value is smaller than the minimum address for data that is registered in the absolute module file, the free-area set data is additionally output to the specified address range.
- An error results if the start address and the end address values are the same.

Description example

```
>lmc30 -F FF sample
```

Data "00H" is output to a free area within the specified address range starting from 1000H and ending with 11FFH.

```
>lmc30 –A 1000:11FF -F 00:1000:10FF sample
```

Data "00H" is output to a free area from address 1000H to address 10FFH within the specified address range starting from 1000H and ending with 11FFH.

```
>lmc30 -F 00:1000:11FF sample
```

Data "00H" is output to a free area from address 1000H to address 11FFH. If the data area registered in the absolute module file sample does not exist within addresses 1000H through 11FFH, the data registered in sample and data "00H" are output to addresses 1000H through 11FFH.

```
>lmc30 -F 00:1000 sample
```

Data "00H" is output to a free area in an address range starting from 1000H and ending with the last data address registered in the absolute module file sample. An error results if the last data address registered in the absolute module file sample is smaller than 1000H.

-H

Converts File into Intel HEX Format

Function

- The lmc30 generates an Intel HEX format file.
- The lmc30 generates an Original HEX format for 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

```
>lmc30 -H debug
```

-ID

Set ID code for ID check Function

Function

- For details on the ID code check, see the hardware manual of the microcomputer.
- The specified ID code is stored as 8-bit data in ID store addresses. And FF is stored in ROM code protect control address or option function select register.

option	Address for ID Code Stored
Non	FFFDF,FFFE3,FFFEB,FFFEF,FFFF3,FFFF7,FFFFB
-R8C	FFDF,FFE3,FFEB,FFEF,FFF3,FFF7,FFFB

- When options (-protect1, -protectx, -ofsregx) to use ROM code protect function or option function select register is specified, the following protect code is filled in protect code store address or register store address.

-ID	-protect1, -protectx, -ofsregx	code
Specify	Specify	Option setting value
Specify	Non	FF
Non	Specify	Option setting value
Non	Non	Value filling in source program

- 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 FFFFFFFFFFFFFF.
- 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.

Precaution

When assembler directive command ".ID", “.OFSREG” or ".PROTECT" is described, this option isn't processed.

Description rule

- 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	FFFDF	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: FFFFFFFFFFFFFF

-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

```
>lmc30 -L debug
```

-O

Specifies Output File Name

Function

- Specify the file name of the machine language file generated by lmc30.
- A path can be specified in the file name.
- The extension of the file name can be specified. A default extension is used for the generated file: ".mot" for the Motorola S format and ".hex" for the Intel HEX format.
- An output file is output in the directory which is the same as the specified absolute module file.

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.

Description example

```
>lmc30 -O test debug
```

A "test.mot" file is generated.

```
>lmc30 -O tmp\test debug
```

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

-ofsregx

Set the option function select register

Function

- For details on the option function select register, see the hardware manual of the microcomputer.
- A specified value is stored in the option function select register.
- If you filled in the option function select register with value, when this option is specified, the data is changed. When this option is not specified, filling value is output.

Precaution

When assembler directive command ".ID", ".OFSREG" or ".PROTECT" is described, this option isn't processed.

Description rule

- Always specify this command option in small letters.
- Specify this option either in the form of "-ofsregx (value).
- Put at least one space between this option and the data.
- Be sure to give an address in hexadecimal.
- The range of 0 to FFH can be written in the operand.
- Always specify this option combining "-R8C" option.
- "-protect1" and "-protectx" option cannot be specified at the same time as this option.

Description example

```
>lmc30 –ofsregx FF debug
```

-protect1

Set levet1 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 ROM code protect control address.
- 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.

Precaution

When assembler directive command ".ID", ".OFSREG" or ".PROTECT" is described, this option isn't processed.

Description rule

- Always specify this command option in small letters.
- "-protectx" option cannot be specified at the same time as this option.
- "-R8C" and "-ofsregx" option cannot be specified at the same time as this option.

Description example

```
>lmc30 -protect1 sample
```

-protectx

Set the ROM code protect control address

Function

- For details on the ROM code protect function, see the hardware manual of the microcomputer.
- A specified value is stored in ROM code protect control address.
- 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.

Precaution

When assembler directive command ".ID", ".OFSREG" or ".PROTECT" is described, this option isn't processed.

Description rule

- Always specify this command option in small letters.
- "-protect1" option cannot be specified at the same time as this option.
- "-ofsregx" option cannot be specified at the same time as this option.
- Specify this option either in the form of "-protectx (protect code).
- Put at least one space between this option and the data.
- Be sure to give an address in hexadecimal.
- The range of 0 to FFH can be written in the operand.

Precaution

When "-R8C" option is specified, it is processed as "-ofsregx".

Description example

```
>lmc30 –protectx FF debug
```

-R8C/-R8C

Generates code that conforms to the R8C Family

Function

- Generates a code that conforms to the R8C Family.

Option	Address area
-R8C	0H to 0FFFFH
-R8CE	0H to 0FFFFFH

Description rule

- Specify this option before entering a file name.

Description example

```
>lmc30 –R8C debug
```

-V

Indicates Version Number

Function

- The software indicates the version number of lmc30.

Precaution
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

```
>lmc30 -V
```

Error Messages of lmc30

'-A' Option Illegal format '-A StartAddr:EndAddr
? The start and end addresses are not correctly set.
! Check the start and end addresses.
'-e' option is too long
? The array of -e option parameters is excessively long.
! Check the syntax for this option and rewrite it correctly.
'-F' Option Illegal format '-F Data:StartAddr:EndAddr
? The start and end addresses are not correctly set.
! Check the start and end addresses.
'-protect' or '-ofsreg' option multiple specified
? The option is specified twice or more.
! 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 '-A' option exceed output address
? The specified address is outside the range of data addresses registered in the absolute module file.
! Make sure the address you specify is within the range of data addresses registered in the absolute module file.
Address specified by '-e' option exceed xxx
? The address specified with -e option exceeds xxx.
! Rewrite the address value correctly.
Address specified by '-F' option exceed output address
? The specified address is outside the range of data addresses registered in the absolute module file.
! Make sure the address you specify is within the range of data addresses registered in the absolute module file.
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.
Not enough memory
? Memory is insufficient.
! Increase the memory capacity.
Specifies option that can't use in M16C
? An invalid option 'option' is specified.
! Specify the option correctly again.
Specifies option that can't use with '-R8C'
? '-R8C' option cannot be specified at the same time as this option.
! Specify the option correctly again.

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

Warning Messages of lmc30

'-ID' option isn't processed.
? '-ID' option isn't processed..
! Assembler directive command ".ID" or ".PROTECT" is described.
‘-ofsreg’ option isn’t processed
? '-ofsreg' option isn't processed.
! Assembler directive command ".ID" or ".PROTECT" is described.
‘-protect’ or ‘-ofsreg’ option isn’t processed
? '-protect' or '-ofsreg' option isn't processed.
! Assembler directive command ".ID", ".OFSREG" or ".PROTECT" is described.
'filename' does not contain object data
? The specified file does not contain object data.
! Check the file name.
Address exceed xxx
? The address exceeded xxx.
! Check the written contents of the source program. Check to see how sections are located.
Original HEX format for microcomputers is generated
? Microcomputers exclusive use file was generated.
! Confirm that microcomputers exclusive use file is not in the problem.

Method for Operating lb30

This section explains the method for operating lb30 to utilize its functions. The primary function of lb30 is to manage multiple relocatable module files as a single library file.

Command Parameters

The table below lists the command parameters available for lb30.

Parameter name	Function
File name	Relocatable module file name to be processed by lb30.
-.	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 lb30.

Order in which to specify command parameters

Always specify the command parameters for lb30 in the following order. If the command parameters are specified in an incorrect order, lb30 cannot process files correctly.

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

```
lb30 (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, lb30 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 ln30 for details on how to specify a command file.

Command Options of lb30

The following pages explain the rules for specifying the command options of lb30.

-.

Disables Message Output to Screen

Function

- The software does not output messages when lb30 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

```
>lb30 -. -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, lb30 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, lb30 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, lb30 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

```
>lb30 -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.

Precaution

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

```
>lb30 -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

```
>lb30 -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

```
>lb30 -L new
```

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

```
>lb30 -L new sample1
```

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

```
>lb30 -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

```
>lb30 -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

```
>lb30 -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 lb30 to the screen.

Precaution

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

```
>lb30 -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

```
>lb30 -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 lb30.

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

```
>lb30 @cmdfile
```

Error Messages of lb30

'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 lb30.
'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 as30.
'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.
'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
? The number of characters per line in the command file exceeds the limit.
! 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.
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 lb30

'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 xrf30

This section explains the method for operating xrf30 to utilize its functions. The basic function of xrf30 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 xrf30.

Parameter name	Function
File name	Source or assembler list file name to be processed by xrf30.
-.	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 xrf30.

Order in which to specify command parameters

The command parameters of xrf30 can be specified in any order.

```
>xrf30 (file name) (command option)
>xrf30 (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 xrf30 referencer allows you to specify a command file name that contains input parameters.
- Refer to the Method for Operating ln30 for details on how to specify a command file.

Command Options of xrf30

The following pages explain the rules for specifying the command options of xrf30.

-.

Disables Message Output to Screen

Function

- The software does not output messages when xrf30 is processing.
- Error messages are output to screen.

Description rule

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

Description example

```
>xrf30 -. 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

```
>xrf30 -N sample.lst
```

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

```
>xrf30 -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

```
>xrf30 -O\work\list sample.a30
```

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

```
>xrf30 -O\work\list sample.lst
```

-V

Indicates Version Number

Function

- The software indicates the version number of the cross referencer.

Precaution

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

```
>xrf30 -V
```

@

Specifies Command File

Function

- The software uses the contents of a specified file as command parameters as it invokes xrf30.

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

```
>xrf30 @cmdfile
```

Error Messages of xrf30

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
? The number of characters per line in the command file exceeds the limit.
! 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 abs30

This section explains the method for operating abs30 to utilize its functions. The primary function of abs30 is to generate an absolute list file from a specified assembler list file.

Precautions using abs30

- 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 as30 processed the source file that contains macro directive com-mand.
  Specify command option "-LS" when the assembler as30 processed the source file that contains structured directive command for AS30.
- It is needed that header lines are output to assembler list file. Operate as30 without command option -H.

Command Parameters

The table below lists the command parameters available for abs30.

Parameter name	Function
File name	Assembler list or absolute modulefile name to be processed by abs30.
-.	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

```
>abs30 (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.

Command Options of abs30

This section describe the rules for specifying the command options of abs30.

-.

Disables Message Output to Screen

Function

- The software does not output messages when xrf30 is processing.
- Error messages are output to screen.

Description rule

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

Description example

```
>abs30 -. 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, abs30 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

```
>abs30 sample -Ddir
```

Assembler list files in "dir" under the current directory are searched.

```
>abs30 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

```
>abs30 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.

Precaution

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

```
>abs30 -V
```

Error Messages of abs30

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 abs30

Address area exceed 0FFFFFH
? The address range exceeds 0FFFFFh.
! 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.
No section information of l-name in x-name
? x-name doesn't have the section information of l-name.
! Absolute-list file can't be generated from l-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 AS30.

Precautions on Writing Program

Pay attention to the following when writing a program to be assembled by AS30:

- Do not use the reserved words of AS30 for names in your source program.
- The character strings consisting of AS30 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 AS30.
- 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 AS30 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 parabraph or line

(Carriage retrn) (Line feed)

Precautions

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

AS30 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.

Precautions

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 Family are the reserved word.

Operators

All operators and structured operators explained in this manual are the reserved word.

Structured directive command

All structured directive commands explained in this manual are the reserved word.

System labels

The system labels are generated by assembler.
AS30 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 as30.

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.

Precautions

You cannot use any name that is identical to one of AS30'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.

Precautions

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
```

Precautions

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 as30 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 as30's directive command is written.
- Only one directive command can be written in one line.
- Comments can be written in the directive command line.

Precautions

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.

Precautions

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 as30 assembler does not process the characters in any line exceeding this limit.

Precautions

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.

Precautions

The as30 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 ('), AS30 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 Family 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 as30.

Function

- Mnemonics available for the M16C family 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.

Precautions

The as30 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 ('), AS30 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.

Precautions

The as30 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 ('), AS30 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.

Precautions

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.

Precautions

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 x 10-38 to 3.40282347 x 1038
DOUBLE (64 bits long): 2.2250738585072014 x 10-308 to 1.7976931348623157 x 10308

Precautions

Floating-point numbers can only be entered for the operands of directive commands ".DOUBLE" and ".FLOAT".

Example:

3.4E35	$3.4*10^{35}$
3.4e-35	$3.4*10^{-35}$
-.5E20	$-0.5*10^{20}$
5e-20	$5.0*10^{-20}$

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.

Precautions
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 as30's source programs.

Precautions
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

Operator	Function
+	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

Operator	Function
+	Adds values on left and right sides of operand together.
-	Subtracts value on right side of operand from value on left side.
*	Multiples 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

Operator	Function
>	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 no 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 no 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

Operator	Function
()	Operation enclosed with ( ) is performed first befor any other operation. If oneexpression contains multiple parentheses, leftmost pair is giben priority. Parenthesezed operations can be nested.

Operation Priority in Expression

The as30 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.

Priority	Type Operator	Operator
1	Operator to change priority (,)	
2	Unary operator	+,-,~,SIZEOF,TOPOF
3	Binary operator 1	*,/,%
4	Binary operator 2	+,-
5	Binary operator 3	>>,<<
6	Binary operator 4	&
7	Binary operator 5	|,^
8	Rlational operator	>,<,>=,<=,==,!=

Expression and Its Value

The following shows a description example of an expression and the value that results from operations performed by as30.

Expression	Result of operation
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

AS30 allows you to write directive commands in addition to the M16C-family mnemonics in the source programs that can be assembled by AS30. 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 as30.

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 as30.

Branch optimization control directive commands

These commands allow you to specify that as30 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-family tool software

These directive commands and operands all are output by the M16C-family tool software.

Precautions

The directive commands output by the M16C-family 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 AS30.

The following pages explains 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.

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

ndicates the end of a conditional assemble block.

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 as30.

@

Concatenates character strings entered before and after @ into a single character string.

.ID

Transfers specified information to map file and ID file.

.INITSCT

Provisionally defines a section name.

.OFSREG

Transfers specified information to map file.

.PROTECT

Transfers specified information to map file.

.RVECTOR

Sets the software interrupt number and software interrupt name.

.SB_AUTO(_xxx)

Automatic Generation of SB Relative Addressing

.SVECTOR

Sets the special page number and special page name.

Control instructions for outputting inspector information

The following are the directive instructions for controlling the output of inspector information.

.INSF

Defines the start of a function (subroutine) in inspector information.

.EINSF

Defines the end of a function (subroutine) in inspector information.

.CALL

Defines where to call a function (subroutine) in inspector information.

.STK

Defines a stack in inspector information.

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

..MACPAR

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 as30.

Function

- This command expands a file name into the one that is being processed by as30 (i.e., assembly source file or include file).

Precautions

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

<sample.a30>

```
.INCLUDE   incfile.inc
.INCLUDE   ..FILE@.inc              ..... (1)
.ASSERT    "comment" > ..FILE       ..... (2)
```

<incfile.inc>

```
.INCLUDE   ..FILE                          ..... (3)
.ASSERT    "comment" > ..FILE@.mes         ..... (4)
```

In the case of above example, they are expanded as follows.

(1) .INCLUDE sample.inc
(2) .ASSERT "comment" > sample
(3) .INCLUDE incfile
(4) .ASSERT "comment" > incfile.mes

In the case of above example, if command option (-F) is specified, the character strings of "..FILE" of (3) and (4) are changed to "sample" not "incfile".

(1) .INCLUDE sample.inc
(2) .ASSERT "comment" > sample
(3) .INCLUDE sample
(4) .ASSERT "comment" > sample.mes

commnad example)

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

..MACPARA

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".

Precautions

If this command is written outside the macro body defined by ".MACRO", its value is made 0.

Description format

..MACPARA

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 repoeat 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".

Precautions

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.

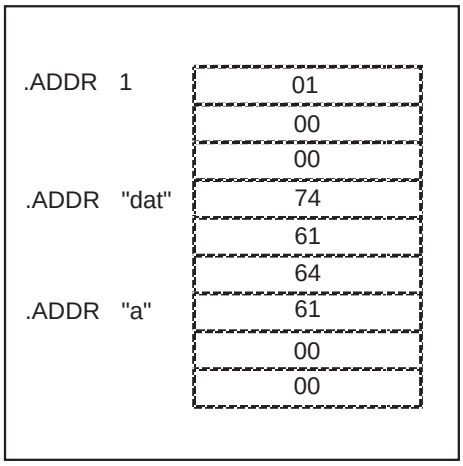
Precautions

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
```



.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
```

.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.
- With an absolute path given to the file name, AS30 generates the file in the given directory.
- With no absolute path given to the file name,
  1. In an instance in which no directory is designated for the file name designated in the command line at the time of starting up AS30:
     AS30 generates the file specified by this command in the current directory.
  2. In an instance in which a directory is designated for a file name designated in the command line at the time of starting up AS30:
     AS30 generates the file with the directory of the file designated in the command line.
- If "..FILE" command is specified as a file name, AS30 generates the file in same directory as the directory of the file designated in the command line at the time of starting up AS30.

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
            .SECTIONarea,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.

Precautions

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.

Precautions

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.

Precautions

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

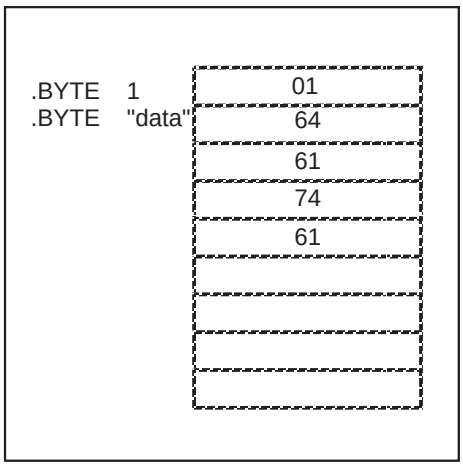
```
                .BYTE     (numeric value)
(name:)         .BYTE     (numeric value)
```

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
```



.CALL

Defines where to call a Function in inspector information

Function

- Defines where to call a Function (subroutine) in inspector information.

Format

CALL (called Function (subroutine) name), (storage class)

Description rules

- Be sure to put a space or a tab between this directive instruction and the operand.
- Be sure to give a called Function (subroutine) name and a storage class.
- Separate a storage class by use of a comma.
- As for the storage class, give either G (global label) or S (local label).

Notes

Give this directive instruction within a range from the start of a Function in inspector information to its end.

This directive instruction turns effective when the command option -finfo has been chosen.

Description example

```
.INSF glbfunc, G, 0
:
jsr glbsub
.CALL glbsub, G
:
jsr locsub
.CALL locsub, S
:
.EINSF
```

.DEFINE

Defines string symbol

Function

- This command defines a character string to a symbol.
- A symbol can be redefined.

Precautions

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
```

.EINSF

Defines the end of a Function in inspector information

Function

- Defines the end of a Function (subroutine) in inspector information.
- Defines the extent from .INSF to the end of a Function (subroutine) as a single function (subroutine).

Format

.EINSF

Description rules

- In using this directive instruction, be sure to use the directive instruction .INSF.
- This directive instruction is for exclusive use with the assembly language. Using this directive instruction in the asm Function in NC30 results in an error.
- This directive instruction turns effective when the command option -finfo has been chosen.

Description example

```
.INSF glbfunc, G, 0
:
.EINSF
```

.ELIF

Rexolves 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 biginnig of ablock 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.

Precautions

The as30 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 (-2147483648 to 2147483647) 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.

Precautions

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 0FFFFFH can be written in the operand.

Precautions

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.

Precautions

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
```

.ID

Set ID code for ID check Function

Function

- The specified ID code is stored as 8-bit data in ID store addresses. And FF is stored in ROM code protect control address or option function select register.
- The specified value is output the map file and ID file.
- The value set to ID code is output also to the absolute module file (.x30).

Precaution

For details on the ID code check, see the hardware manual of the microcomputer.

Description format

.ID "(ID code character string)"
.ID “#(ID code numeric value)”

Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- The specified ID code is stored as an ASCII code.
- Specify ID code character string within 7 letter.
- Specify after the ID code value adds '#' to the head.
- ID code value is stored as the numerical value.
- Specify ID code value by the integer value within 14 digits.
- This directive command can be described in the assembly sauce file only in 1 degree.

Precautions

When this directive command is described in more than one assembly sauce file, it becomes warning with the linkage editor.

Description example(ID code character string)

```
; fixed vector section
;------------------------------------------------------------
      .org    0FFFCh
RESET:
      .lword     start
      .id    “Code”    ; Sets ID code "Code"
```

Description example(ID code numeric value)

```
    . ; fixed vector section
;------------------------------------------------------------
      .org    0FFFCh
RESET:
      .lword     start
.id    “#20030401”       ; Sets ID code "20030401"
```

.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 as30 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.

Relational operators	Contents
>	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.

Precautions

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.

Precautions

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
'smp1'==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.
- With either an absolute path given to the name of a file to include, AS30 searches the given directory for the file. If the file is not found, an error occurs.
- With either an absolute path or a relative path given to the name of a file to include, AS30 searches the given directory for the file. If the file is not found, an error occurs.
- With no path included in the name of a file to include, AS30 searches for the file in the sequence given below:
  1 In an instance in which no directory is designated for the file name designated in the command line at the time of starting up AS30:
  AS30 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 AS30:
  AS30 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 AS30 searches the directory designated by the command option -I.
  3 AS30 searches the directory set in the environment variable INC30.

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.
- Nesting level of include files is within 9.

Precautions

Do not specify INCLUDE the file itself within the include file.

Description example

```
.INCLUDE    initial.a30
.INCLUDE.   .FILE@.inc
```

.INITSCT

Provisionally defines a section name.

Function

- Provisionally defines a section name.
- This is a C language startup-only directive command.

Precautions

This directive command is generated by the C language startup (initsct.c) initialization function, and is usable in only a compiler.

Format

.INITSCT (section name), (section type), align
.INITSCT (section name), (section type), noalign

Description rules

- Be sure to put a space or a tab between this directive instruction and the operand.
- Separate a section name and a section type by use of a comma.
- Separate a section type and align or noalign by use of a comma.
- For the section type, write ‘CODE,’ ‘ROMDATA,’ or ‘DATA.’
- If *align* is specified, the address in which to store the code for the line that immediately follows this directive command is corrected to be 2, 4 or 8-byte aligned.

Description example

```
.initsct   bss_NE, data, align          ;get alignment
.initsct   bss_NO, data, noalign        ;not get alignment
        :
```

.INSF

Defines the start of a Function in inspector function

Function

- Defines the start of a function (subroutine) in inspector information.
- Defines the extent from the start of a Function (subroutine) to the directive instruction .EINSF as a single function (subroutine).

Format

.INSF (Function (subroutine) start label), (storage class), (frame size)

Description rules

- Be sure to put a space or a tab between this directive instruction and the operand.
- Be sure to give a Function (subroutine) start label, a storage class, and a frame size.
- Separate a storage class and a frame size by use of a comma.
- As for the storage class, give either G (global label) or S (local label).
- Use an integer to give a frame size.

Notes

In using this directive instruction, be sure to use the directive instruction .EINSF.
This directive instruction is for exclusive use with the assembler language. Using this directive instruction in the asm Function in NC30 results in an error.
This directive instruction turns effective when the command option -finfo has been chosen.

Description example

```
glbfunc:
    .INSF   glbfunc, G, 0
        :
    .EINSF

locfunc:
    .INSF   locfunc, S, 0
        :
    .EINSF
```

.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.

Precautions
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.

Precautions
Kanji and other 8-bit code are not processed correctly. However, the as30 assembler does not output errors.

- Always be sure to enclose the character string with quotations as you write it.

Precautions
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 used to write a character string.

Precautions
Kanji and other 8-bit code are not processed correctly. However, the as30 assembler does not output errors.

- Always be sure to enclose the character string with quotations as you write it.

Precautions
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

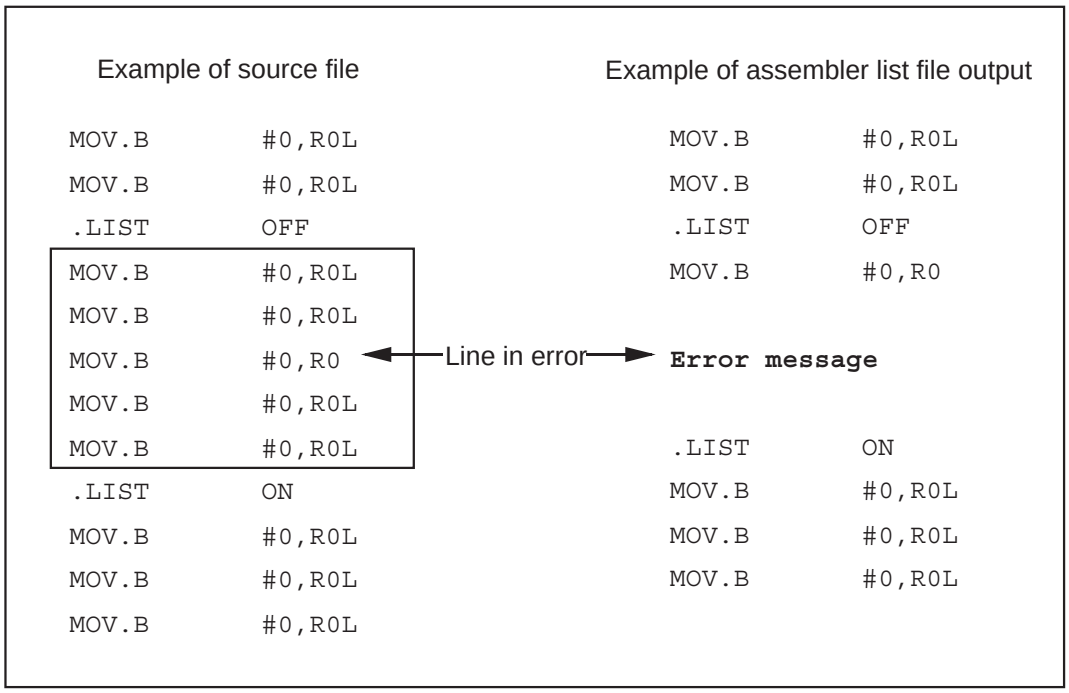
.LIST [ON|OFF]

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
```



.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.

Precautions

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.

Precautions

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.

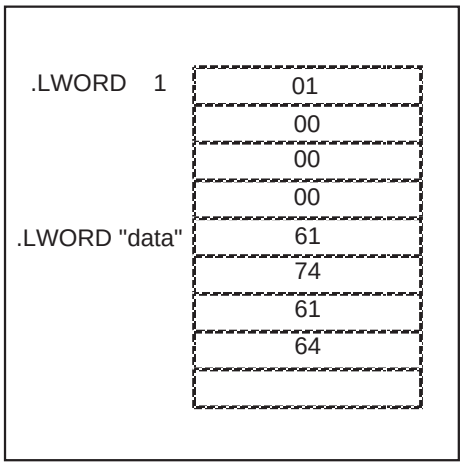
Precautions

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.

Precautions
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.

Precautions
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.

Precautions

If the number of actual parameters does not match that of formal parameters, the as30 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.

Precautions

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
```

.OFSREG

Set the option function select register

Function

- The specified value is stored in option function select register.
- The specified value is output the map file.
- The value set to the option function select register is output also to the absolute module file (.x30).

Precaution

For details on the option function select register, see the hardware manual of the microcomputer.

Description format

.OFSREG (numeric value)”

Rules for writing command

- Use this option in combination with “-R8C” option.
- Always be sure to insert space or tab between the directive command and the operand.
- AN integer in the range of 0 to 0FFH can be written in the operand.
- A symbol can be written in the operand.
- This directive command can be described in the assembly sauce file only in 1 degree.

Precautions

When this directive command is described in more than one assembly sauce file, it becomes warning with the linkage editor.

When “-R8C” option is not specified, it is processed as ".PROTECT".

Description example

```
    . ; fixed vector section
;------------------------------------------------------------
      .org    0FFFCh
RESET:
      .lword     start
.ofsreg    0FFH       ; Sets the option function select register 0FFH
```

.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.

Precaution

When the nc30 “-OGJ(-Oglb_jmp)” or as30 “-JOPT” option was specified, this directive command is disregarded.

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:

Kind of parameter	Parameter	Function
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.

Precautions

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.

Precautions

If directive command ".ORG" is not found in the line immediately following description of ".SECTION", the section is assigned relative attribute.

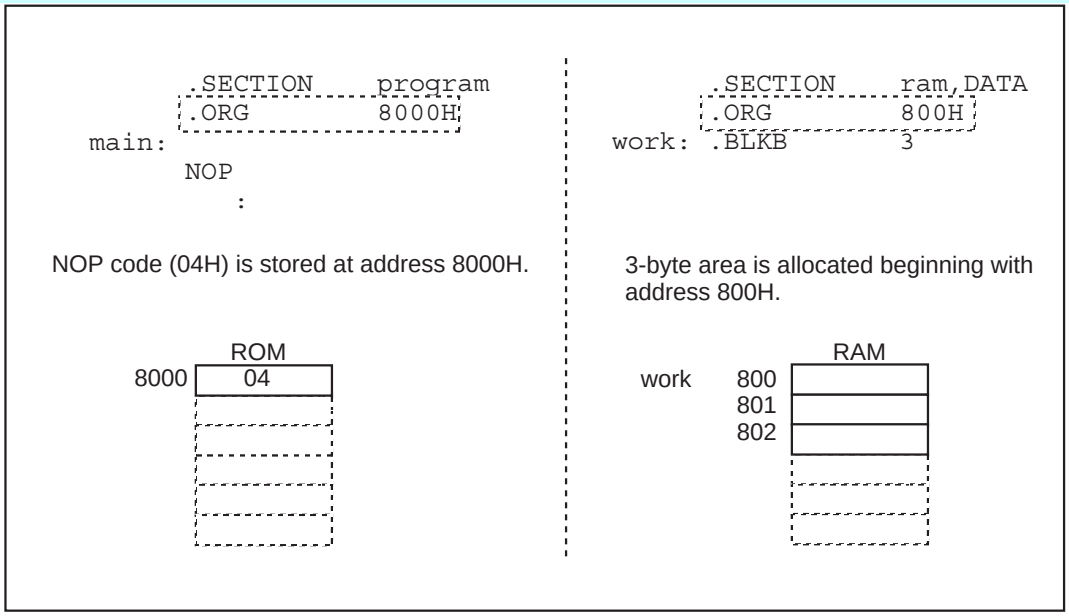
- This directive command cannot be written in relative-attribute sections.
- Always be sure to insert space or tab between the directive command and the operand.
- The values that can be written in the operand are a numeric value in the range of 0 to 0FFFFFH.
- An expression can be written in the operand. 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      "abcdefghijklmnopqrstuvwxyz"
.ORG       0FF80H
.BYTE      "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

The following statement results in an error.

```
.SECTION   value,ROMDATA
.BYTE      "abcdefghijklmnopqrstuvwxyz"
.ORG       0FF80H
.BYTE      "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```



.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.

Precautions

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'
```

.PROTECT

Set the ROM code protect control address

Function

- The specified value is stored in ROM code protect control address.
- The specified value is output the map file.
- The value set to the ROM code protection control address is output also to the absolute module file (.x30).

Precaution

For details on the ROM code protect function, see the hardware manual of the microcomputer.

Description format

.PROTECT (numeric value)”

Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- AN integer in the range of 0 to 0FFH can be written in the operand.
- A symbol can be written in the operand.
- This directive command can be described in the assembly sauce file only in 1 degree.

Precautions

When this directive command is described in more than one assembly sauce file, it becomes warning with the linkage editor.

Description example

```
    . ; fixed vector section
;------------------------------------------------------------
      .org    0FFFCh
RESET:
      .lword     start
.protect    0FFH       ; Sets protect code 0FFH
```

.RVECTOR

Set the software interrupt

Function

- The software interrupt number and software interrupt name are set.
- The contents set to this directive command are allocated to the variable vector table.
- The content set in this directive command is output to a map file generated by the linker as variable vector table information.

Precaution

- If this directive command is written, a variable vector table is automatically generated by the linker. If as a result of automatic generation, a free space is created in the variable page vector table (any software interrupt not specified by this directive command), a value for the free space is set in order of the priority given below.
  (1) Value set by the link option "-VECT"
  (2) Value of a global label "__dummy_int"
  (3) Value of a global label "dummy_int"
  Note that if any one of the values in (1) to (3) is not set, no values are set in the free space.
- If this directive command is written, 1,024 bytes of area is reserved for use as the section name 'vector.'
- If a program is written in the "vector" section while this directive command is written, this directive command results in an error. (Do not write a program in the "vector" section.)
- The software interrupt numbers specified by this directive command cannot be specified in the link option "-VECTN."

Description format

.RVECTOR software interrupt No., software interrupt name

Rules for writing command

- Be sure to describe the space or tab between the directive command and operand.
- Be sure to describe the comma between the software interrupt number and software interrupt name.
- For software interrupt No., only the value defined in assembling can be described.
- Software interrupt No. can be described in the range of "0 to 63".
- For the software interrupt name, the symbol or label can be described.

Description example

```
.rvector   21, timerA0        ;Sets timerA0 to software interrupt No.21.
```

.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.

Precautions

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 SB relative displacement addressing mode is selected for the name that is specified in the operand of this directive command.
- If the 1-bit manipulate command takes on a short format, a 11-bit SB relative displacement or 16-bit relative displacement addressing mode is selected.
- If the 1-bit manipulate command does not have a short format, an 8-bit SB relative displacement or 16-bit relative displacement addressing mode is selected.

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

```
.SB       80H
LDC       #80H,SB
.SBBIT    bitsym
BCLR      bitsym     ; Selects a 11-bit SB relative.
BAND      bitsym     ; Selects a 16-bit SB relative.
```

.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.

Precautions

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
```

.SB_AUTO

.SB_AUTO / .SB_AUTO_S /.SB_AUTO_SBVAL / .SB_AUTO_SBSYM / .SB_AUTO_R / .SB_AUTO_E

Automatic Generation of SB Relative Addressing

Function

- The assembler selects the SB relative addressing mode.
- Generates the instructions to save/restore SB register and to set register values.

.SB_AUTO	Shows that automatic generation of SB relative addressing will start.
.SB_AUTO_S	Shows the beginning of the function.
.SB_AUTO_SBVAL	Generates the instruction to save SB register (PUSHC) and the instruction to set register values (LDC).
.SB_AUTO_SBSYM	Selects the SB relative addressing mode for the name specified in the operand.
.SB_AUTO_R	Generates the instruction to restore SB register (POPC).
.SB_AUTO_E	Shows the end of the function.

Precautions

- These directive commands are C complier-only directives, so that they cannot be written in user programs.
- Depending on condition, no instructions will be generated by .SB_AUTO_SBVAL and .SB_AUTO_R.

Description format

```
.SB_AUTO
.SB_AUTO_S          C language function name, assembler function name
.SB_AUTO_SBVAL      SB register set value
.SB_AUTO_SBSYM      SB relative addressing target symbol
.SB_AUTO_R
.SB_AUTO_E
```

Example of compiler output

```
        .glb    _func1
_func1:
        .sb_auto_s          func1,_func1
        .sb_auto_sbval      _i1
        .sb_auto_sbsym      _i1,_i2,_i3
          ;
        .sb_auto_r
        rts
        .sb_auto_e
```

.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, ln30 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, as30 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:
    :
```

.STK

Defines a stack in inspector information

Function

- Defines a stack in inspector information.

Format

.STK stack size

Description rules

- Be sure to put a space or a tab between this directive instruction and the operand.
- Be sure to give a stack size.
- Use an integer to give a stack size.

Notes

Use this directive instruction within a range from the start of a function in inspector information to its end.

This directive instruction turns effective when the command option -finfo has been chosen.

Description example

```
.INSF glbfunc, G, 0
:
.STK 2          ;2-byte push
jsr glbsub
.STK -2         ;2-byte pop
:
.EINSF
```

.SUBSTR

Extracts specified number of characters

Function

- This command extracts a specified number of characters from the specified position of a character string.

Precautions

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.

Precautions

Kanji and other 8-bit code are not processed correctly. However, the as30 assembler does not output errors.

- Always be sure to enclose the character string with quotations as you write it.

Precautions

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.

.SVECTOR

Sets the special page

Function

- The special page number and special page name are set.
- The contents set to this directive command are allocated to the special page vector table.
- The content set in this directive command is output to a map file generated by the linker as special page vector table information.

Precaution

- If this directive command is written, a special page vector table is automatically generated by the linker.
- If this directive command is written, a section name “svector” is defined.
- When linked, the section name “svector” has a memory area reserved for it from special page number 18 up to the largest special page number that is specified.
- If as a result of automatic generation, a free space is created in the special page vector table (any special page number not specified by this directive command), the free space is filled with FFH.
- If a program is written in the “svector” section while this directive command is written, this directive command results in an error. (Do not write a program in the “svector” section.)

Description format

.SVECTOR special page No., special page name

Rules for writing command

- Be sure to describe the space or tab between the directive command and operand.
- Be sure to describe the comma between the special page number and special page name.
- For special page No., only the value defined in assembling can be described.
- Special page No. can be described in the range of “18 to 255”.
- For the special page name, the symbol or label can be described.

Description example

```
.svector  250, __SPECIAL_250      ;Sets __SPECIAL_250 to special page No.250.
```

.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 ln30.
- 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.

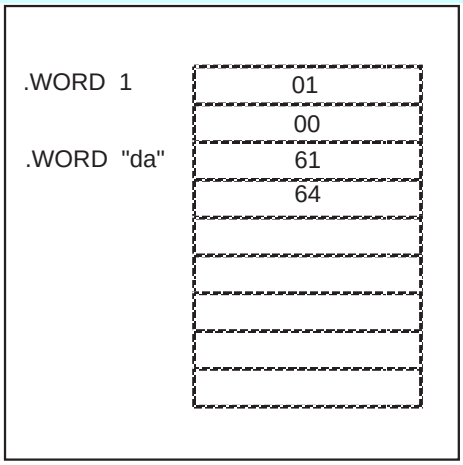
Precautions

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.

Precautions

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 "tl0001","tl0002" ... and "tlFFFF".

Description format

```
?:
    (mnemonic)      ?+
    (mnemonic)      ?-
```

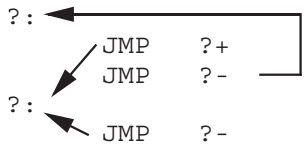
Rules for writing command

- 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.



@

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.

Precautions

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

If the currently processed file name is "sample1.a30", a message is output to the sample.dat file.

```
.ASSERT     "sample" > ..FILE@.dat
```

- A macro definition like the one shown below can be entered:

```
mov_nibble  .MACRO              p1,src,p2,dest
     MOV@p1@p2      src,dest
    .ENDM
                :
    mov_nibble          L,R0L,H,[A0]
                :
    MOVLH   R0L,[A0]
```

Structured Description Function

Outline

Programming with AS30 allows you to enter structured descriptions using structured description commands.

The following lists the functions of AS30's structured description:

- The assembler generates assembly language branch instructions corresponding to the structured description commands.
- The assembler generates labels indicating the jump address for the generated branch instruction.
- The assembler outputs the assembly languages generated from the structure description commands to an assembler list file. (When a command option is specified)
- A structured description command allows you to select a control block that is made to branch by a structured description statement and its conditional expression. A control block refers to a program section from some structured description statement to the next structured description statement except for assignment statements.

Structured Description Statement

The following outlines a structured description statement.

Types of Structured Description Statements

AS30 allows you to write the following nine types of statements:

Assignment statement

The left side is substituted for by the right side.

IF ELIF ELSE ENDIF statement (hereafter called the IF statement)

The IF statement changes the flow of control to two directions. The direction in which control branches is determined by a conditional expression.

FOR NEXT statement (hereafter called the FOR-NEXT statement)

The FOR-NEXT statement controls repetition of operation. The statement is executed repeatedly as long as the specified conditional expression is true.

FOR TO STEP NEXT statement (hereafter called the FOR- STEP statement)

The FOR-STEP statement controls the number of repetitions by specifying the initial value, an increment, and the final value.

DO WHILE statement (hereafter called the DO statement)

The DO statement executes the statement repeatedly as long as the conditional expression is met (true).

SWITCH CASE DEFAULT ENDS statement (hereafter called the SWITCH statement)

The SWITCH statement causes control to branch to one of the CASE blocks depending on the value of the conditional expression.

BREAK statement

This statement causes the relevant FOR, DO, or SWITCH statement to stop executing, transferring control to the statement to be executed next.

CONTINUE statement

This statement transfers control to a statement in the least repeated FOR or DO statement that determines repetition.

FOREVER statement

This statement executes the control block repeatedly assuming that the conditional expression of the relevant FOR and DO statements are always true.

Types of Variables

In AS30's structured description, the microcomputer registers and memories are referred to as variables. There are following types of variables:

Register variable

This refers to the registers in the M16C family microcomputers.

Flag variable

This refers to the function flags of the M16C family.

Register bit variable

This refers to each bit position of a register variable.

Memory variable

This refers to an arbitrary label or symbol.

Memory bit variable

This refers to an arbitrary bit symbol.
Details on how to write each variable are explained in other sections of this manual.

Reserved Variables

In AS30's structured description, the register, flag, and register bit variables are processed as reserved variable names. Therefore, you cannot use a memory variable name or symbol name for the names used in these variables. For details about the register and flag functions, refer to the "M16C Family Software Manual."

Register Variables

The table below lists the register variables available for the M16C family. The as30 assembler does not discriminate register variable names between uppercase and lowercase letters. Consequently, "R0L" and "r0l" refer to the same register variable.

Variable Name	Register Name	Variable Type Name
R0L,R0H,R1L,R1H	Data register	Byte type
R0,R1,R2,R3	Data register	Word type
A0.B,A1B	Address register	Byte type
A0,A0.W,A1,A1.W	Address register	Word type
[A0.B],[A1.B]	Address register indirect	Byte type
[A0],[A0.W],[A1],[A1W]	Address register indirect	Byte type
[A0.A],[A1.A]	Address register indirect	Address type
[A0.L],[A1L]	Address register indirect	Long word type
FB	Frame base register	Word type
PC	Program counter	Address type
INTBH,INTBL	Interrupt table register	Word type
INTB	Interrupt table register	Address type
SP,ISP	Stack pointer	Word type
SB	Static base register	Word type
FLG	Flag register	
R2R0,R3R1	32-bit data register	Long word type
A1A0	32-bit address register	Long word type
[A1A0.B]	32-bit address register indirect	Byte type
[A1A0],[A1A0.W]	32-bit address register indirect	Word type
IPL	Processor interrupt priority level	

Precautions

SP refers to the stack pointer (user stack pointer or interrupt stack pointer) indicated by the U flag. For details about the stack pointer and U flag functions, refer to the "M16C Family Software Manual."

Stack Variables

The table below lists the stack variables available for the M16C family. The as30 assembler does not discriminate variable names between uppercase and lowercase letters. Consequently, "STK" and "stk" refer to the same variable.

Stack Variable Name	Content
[STK]	Memory indicated by stack pointer.

Stack variables can be written for saving or restoring to or from the stack area.

Precautions
The stack area is indicated by the interrupt stack pointer when the U flag = 0 or the user stack pointer when the U flag = 1.

Flag Variables

The table below lists the flag variables available for the M16C family. The as30 assembler does not discriminate flag variable names between uppercase and lowercase letters. Consequently, "C" and "c" refer to the same flag variable. For details about the functions of flag variables, refer to the "M16C Family Software Manual."

Flag Variable Name	Flag Name
C	Carry flag
D	Debug flag
Z	Zero flag
S	Sign flag
B	Register bank specifying flag
O	Overflow flag
I	Interrupt enable flag
U	Stack pointer specifying flag

Register Bit Variables

The table below lists the register bit variables available for the M16C family. The as30 assembler does not discriminate register bit variable names between uppercase and lowercase letters. Consequently, "BITR0_1" and "bitr0_1" refer to the same register bit variable.

Register Bit	Variable Name Content
BITR0_n	Bit n of data register R0 (n = 0 to 15)
BITR1_n	Bit n of data register R1 (n = 0 to 15)
BITR2_n	Bit n of data register R2 (n = 0 to 15)
BITR3_n	Bit n of data register R3 (n = 0 to 15)
BITA0_n	Bit n of data register A0 (n = 0 to 15)
BITA1_n	Bit n of data register A1 (n = 0 to 15)

Register bit variable description example

The following shows a description example for register bit variables:

```
BITR0_0 = 0    ; Substitutes 0 for the 0th bit of register R0.
BITR1_1 = 0    ; Substitutes 0 for the 1st bit of register R1.
BITR2_2 = 0    ; Substitutes 0 for the 2nd bit of register R2.
BITR3_3 = 0    ; Substitutes 0 for the 3rd bit of register R3.
BITA0_4 = 0    ; Substitutes 0 for the 4th bit of register A0.
BITA1_5 = 0    ; Substitutes 0 for the 5th bit of register A1.
;
IF BITR0_1     ; Evaluates the 1st bit of register R0.
   :
ELSE
   :
ENDIF
;
IF BITR0_2     ; Evaluates the 2nd bit of register R0.
   :
ELSE
   :
ENDIF
```

Memory Variables

In AS30's structured description, labels and symbols are processed as memory variables.

The as30 assembler discriminates memory variable names between uppercase and lowercase letters.

Types of Memory Variables

The label and symbol names defined by the directive commands listed in the table below can be used in structured description statements as memory variables. The variable has its "variable type" defined by the directive command.

Assembler Directive Command	Variable Type
.BTEQU, .BTGLB	Bit type
.BLKB, .BYTE	Byte type
.BLKW, .WORD	Word type
.BLKA, .ADDR	Address
.BLKL, .LWORD	Long word type
.GLB	For externally referenced labels and symbols, write the size every line or use a command option to determine the size.

The assembler generates object code according to the variable type.

Function of command option '-M'

- If the type of variable is not indicated when as30's command option '-M' is specified, the assembler assumes the byte type as it generates object code.
- If this command option is not specified, the assembler assumes the word type as it generates object code.

Memory Variable Addressing Modes

The table below lists the address modes that can be specified in memory variables:

Addressing Mode	Addressing Mode Description Format
Absolute	[label:16] [label:20]
Address register relative	[label:8[A0]] [label:16[A0]] [label:8[A1]] [label:16[A1]] [label:20[A0]]
SB relative	[label:8[SB]] [label:16[SB]]
FB relative	[label:8[FB]]

The addressing mode specifier (:8, :16, or :20) can be omitted.

Rules for Writing Memory Variables

- When writing a memory variable name in structured description statement, always be sure to enclose it with brackets [ ] or { } as you write it
- A space or tab can be entered between the memory variable name and brackets.
- When specifying an addressing mode, always be sure to enclose it with brackets [ ] or { } along with the variable name as you write it.

Description example 1:

```
    .GLB        work
    .SECTION    memory,DATA
mem:            .BLKB       1
    .SECTION    program,CODE
    [mem] = 0
    [work].B = 0
    .END
Description example 2:
    [ label ] = 10
Description example 3:
    IF [ label[SB] ]
                :
    ELSE
                :
    ENDIF
```

Size Specifier

The size specifier can be set for memory variables and address register indirect addressing [A0] or [A1]. For memory variables where a size specifier is written, the assembler temporarily generates code in the specified size irrespective of the type of variable that is determined when defining a memory variable.

The table below lists the size specifiers that can be written in memory variables.

Size Specifier	Variable Type
.B	Byte type
.W	Word type
.A	Address type
.L	Long word type

Precautions

The type of memory variable on a line where a size specifier is set has priority over the type determined by a directive command.

Rules for Writing Size Specifier

- Write a size specifier immediately after the memory variable that is enclosed with brackets.
- A space or tab can be entered between the size specifier and brackets.

Description example:

```
                .SECTION  ram,DATA
lab_b:          .BLKB     1
lab_w:          .BLKW     1
                :
                .SECTION  rom,CODE
                :                               ;Example of Expansion
[ lab_b ] = R0L                 ;MOV.B    R0L,lab_b
[ lab_b ].W = R0                ;MOV.W    R0,lab_b
[ lab_w ] = R0                  ;MOV.W    R0,lab_w
[ lab_w ].B = R0L               ;MOV.B    R0L,lab_w
```

Memory Bit Variables

The bit symbol names defined by the directive commands listed below can be used in structured description statements as a memory bit variable.

Assembler Directive Command	Variable Type
.BTEQU, .BTGLB	Bit type

Memory Bit Variable Addressing Modes

The table below lists the address modes that can be specified in memory bit variables:

Addressing Mode	Addressing Mode Description Format
Absolute	[bitsym:16],[bitnum.addr:16]
SB relative	[bitsym:8[SB]],[bitnum,addr:8[SB]] [bitsym:11[SB]],[bitnum,addr:11[SB]] [bitsym:16[SB]],[bitnum,addr:16[SB]]
FB relative	[bitsym:8[FB]],[bitnum,addr:8[FB]]

The addressing mode specifier (:8, :16, or :20) can be omitted.

In the above table, 'bitnum' denotes a bit number and 'addr' denotes a memory address.

Precautions
Address register indirect and relative addressing cannot be written.

Rules for Writing Memory Bit Variable

- When writing a memory bit variable name in structured description statement, always be sure to enclose it with brackets [ ] or { } as you write it.
- A space or tab can be entered between the memory bit variable name and brackets.
- When specifying an addressing mode, always be sure to enclose it with brackets [ ] or { } along with the variable name as you write it.

Description example 1: For internally defined memory bit variable

```
bitsym      .BTEQU   1,10H     ; Defines a bit symbol.
    IF [ bitsym ]
                :
    ELSE
                :
    ENDIF
```

Description example 2: For externally referenced memory bit variable

```
    .BTGLB   bitsym     ; References a bit symbol.
    IF [ bitsym ]
                :
    ELSE
                :
    ENDIF
```

Structured Operators

The following sections explain the operators that can be written in structured description statements.

Unary Operators

The table below lists the unary operators that can be written in structured description statements.

Operator	Content
+	Represents a positive number.
-	Represents a negative number.
~	Negates every bit. (NOT)
++	Increments a single term.
--	Decrements a single term.

Binary Operators

The table below lists the binary operators that can be written in structured description statements.

Operator	Content
+, +.C, +.D, +.CD	Adds two terms.
-, -.C, -.D, -.CD	Subtracts two terms.
*, *.S	Multiplies two terms.
/, /.S	Divides two terms.
%, %.S, %.SE	Divides two terms with residue.
&	ANDs every bit. (AND)
|	ORs every bit. (OR)
^	Exclusive ORs every bit. (EOR)
>>.C	Bit rotates the left-side value to the right by the right-side value with a carry.
<<.C	Bit rotates the left-side value to the left by the right-side value with a carry.
<>.R	Bit rotates the left-side value by the right-side value without a carry. Rotated left if the right-side value is positive; rotated right if the right-side value is negative.
<>.A	Arithmetically shifts the left-side value for a number of bits indicated by the right-side value. Shifted left if the right-side value is positive; shifted right if the right-side value is negative.
<>.L	Logically shifts the left-side value for a number of bits indicated by the right-side value. Shifted left if the right-side value is positive; shifted right if the right-side value is negative.
&&	Logically ANDs.
||	Logically ORs.

Relational Operators

The table below lists the relational operators that can be written in structured description statements.

Operator	Content
<, <.S	Holds true when the left side is smaller than the right side.
>, >.S	Holds true when the left side is larger than the right side.
==	Holds true when the left and right sides are equal.
!=	Holds true when the left and right sides are not equal.
<=, <=.S	Holds true when the left side is smaller than or equal to the right side.
>=, >=.S	Holds true when the left side is larger than or equal to the right side.

Operator Attributes

The table below lists the operator attributes specified for addition and subtraction of binary operators and in part of relational operators.

Attribute	Meaning
.C	Performs calculation with a carry or borrow.
.D	Performs decimal calculation.
.CD	Performs decimal calculation with a carry or borrow.
.S (except residue)	Performs calculation with a sign.
.S (residue)	The sign of the calculation result is made the same as that of the dividend.
.SE	The sign of the calculation result is made the same as that of the divisor.

Precautions
No space or tab can be entered between the operator and attribute.

Expressions

The following explains expressions that can be written using operators.

Types of expressions

There are following types of expressions:

Monomial expression
: An expression consisting of a single term and an expression consisting of a combination of a single term and unary operator.

Binomial expression
: An expression consisting of two terms and an operator.

Compound expression
: An expression consisting of a combination of a monomial or binomial expression and a logical operator.

Terms in expression

The following can be written in terms of an expression:

Variable
: This includes a register, flag variable, register bit variable, memory variable, and a memory bit variable.

Constant
: For multiplication and residue calculations, the constants shown below can be operated on.

Precautions
Except for binary divide and residue calculations, you cannot write an expression using variables of different types.

Compound expression

The following shows rules for writing a compound expression.

- Up to two logical operators can be written in one expression.
- Operation on a compound expression is performed sequentially from left to right.
- A structured description command and a compound expression must be written in one line not exceeding 255 characters.
- No compound expression can be written in two or more lines.

Example of compound expression:

```
IF [ work1 ] || [ work2 ] && [ work3 ]
   :
ENDIF
```

Example of expression

The following shows examples for each type of expression. In these examples, "mem" and "work" denote memory variable names.

Monomial expression

```
[mem]
-[mem]
++[mem]
```

Binomial expression

```
[mem] + 1
- [mem] + 1
```

Compound expression

```
[mem] || [work]
-- [mem] && [work]
```

Structure of Structured Description Statement

A structured description statement consists of a structured description command and a conditional expression that is written in the operand of the command. Not all structured description commands are accompanied by a conditional expression.

Conditional Expression

Function of conditional expression

- A conditional expression indicates a condition to be given to a structured command statement.
- Depending on whether the operation result of a conditional expression is true or false, the assembler generates object code that causes control to branch to different control blocks.

Rules for writing conditional expression

- A conditional expression can be written in the operand of a structured description command "IF," "ELIF," "FOR (FOR-NEXT)," or "WHILE."
- Expressions can be written in the operand of a conditional expression.
- Always be sure to enter a space or tab between a conditional expression and a structured description command.
- When writing a structured description command and an expression, make sure that they are written in one line (within 255 characters).
- No conditional expression can be written in two or more lines.

Description format

- Expression
- Expression Relational operator Expression
- Bit variable
- Bit variable Relational operator 1
- Bit variable Relational operator 0

Description example

The following shows a description example of a conditional expression. In this example, "mem" and "work" denote memory variable names; "bit" denotes a memory bit variable name.

```
IF [mem]
    :
ENDIF

FOR --[mem]
    :
NEXT

IF [mem] >= 0
    :
ENDIF

FOR [work] - [mem] <= 0
    :
NEXT

IF [bit]
    :
ENDIF

IF [bit] == 1
    :
ENDIF

IF [bit] != 0
    :
ENDIF
```

Nesting of Structured Description Statements

Structured description statements can be nested in up to a total of 65,535 levels. However, no intertwined nesting of statements like the example shown below are accepted.

Furthermore, no intertwined nesting of statements including macro directive commands or assembler directive commands '.IF,' '.ELIF,' '.ELSE,' or '.ENDIF' are accepted.

Example of incorrect (intertwined) nesting

```
FOR R0 = 1 TO 10 STEP 1
      :
   IF   R1 == 3 ;The if statement begins in a for statement.
      :
NEXT
   ENDIF    ; The if statement ends outside the for statement.
```

List of Structured Description Commands

The following pages show rules for writing structured description commands.

IF Statement

The following shows the structure of an IF statement.

IF-ENDIF

Basic structure

- The basic structure of an IF statement consists of structured description commands 'IF' and 'ENDIF' and a control block enclosed with these commands.

```
IF Conditional expression
    Control block
ENDIF
```

Function

- Control branches to ENDIF if the condition of IF is false.
- A system label is generated for ENDIF.

Rules for writing command

- Always be sure to enter a space or tab between 'IF' and the conditional expression.
- Conditional Expression can be used the conditional expression.

ELSE

Function

- Structured description command 'ELSE' can be written in the IF statement.
- If the conditional expression of 'IF' is false, control branches to the control block that follows ELSE.
- If there are two or more control blocks, branching to ENDIF occurs at the end of each control block.
- A system label is generated for ELSE.

Rules for writing command

- Only one instance of 'ELSE' can be written between 'IF' and 'ENDIF.'

```
IF Conditional expression
    Control block
ELSE
    Control block
ENDIF
```

ELIF

Function

- Structured description command 'ELIF' can be written in the IF statement.
- If the conditional expression of IF is false, the assembler checks the conditional expression of ELIF to see if it is true or false.
- A system label is generated for ELIF.
- If the conditional expression of ELIF is true, control branches to the beginning of the immediately following control block.
- If the conditional expression of ELIF is false, control branches to the immediately following structured description command (ELIF, ELSE, or ENDIF).
- If there are two or more control blocks, branching to ENDIF occurs at the end of each control block.

Rules for writing command

- Always be sure to enter a space or tab between ELIF and the conditional expression.
- More than one instance of 'ELIF' can be written between 'IF' and 'ELSE' or between 'IF' and 'ENDIF.'

```
IF Conditional expression
    Control block
ELIF Conditional expression
    Control block
ELSE
    Control block
ENDIF
```

Example of source description

```
IF [ sym1 ] == 10     ; If equal, this line is processed.
    :
ELIF [ sym2 ] != 10 ; Is the value of byte type sym2 not equal to 10?
    :       ; If not equal, this line is processed.
ELSE
    :       ; If neither holds true, this line is processed.
ENDIF
Expansion example
    CMP.B     #10,sym1
    JNE       ..IF0002
                :
    JMP       ..IF0003
..IF0002:
    CMP.B     #10,sym2
    JEQ       ..IF0004
                :
    JMP       ..IF0003
..IF0004:
                :
..IF0003:
```

FOR-STEP Statement

Basic structure

- The basic structure of a FOR statement consists of structured description commands 'FOR' and 'NEXT' and a control block enclosed with these commands.

```
FOR Loop counter = Initial value TO final value [STEP increment]
   Control block
NEXT
```

Function

- The loop counter value specified in the operand of structured description command 'FOR' is updated for a specified amount of increment. When the value becomes equal to the final value, the control block is executed.
- If the loop counter value equals the final value, control branches to the line immediately following structured description command 'NEXT.'
- If the specified increment is a negative value, the loop counter is counted down.

Precautions

The control block is always repeated until the loop counter value becomes equal to the final value.

- If the increment is omitted, the assembler assumes '+1' as it generates object code.
- System labels are generated for 'FOR' and 'NEXT' statement.
- A 'BREAK' statement can be written in the control block. This 'BREAK' statement forcibly terminates repetition control.
- A 'CONTINUE' statement can be written in the control block. This 'CONTINUE' statement causes control to branch to the NEXT statement.

Rules for writing command

- A register variable and memory variable can be written in the loop counter.

Precautions

If the register variable or memory variable used in the loop counter has its content modified in the control block, the FOR statement will not be executed correctly.

- A 'BREAK' statement can be written in any desired line in the control block.
- A 'CONTINUE' statement can be written in any desired line in the control block.
- Variables or constant values can be used in the initial and final values.
- A constant value can be used in the increment.
- A local symbol name can be written as a constant value.

FOREVER

The FOREVER command means that there is no condition to terminate repetition.

```
FOR FOREVER
    Control block
NEXT
```

Function

- This command continues executing the control block repeatedly.
- A 'BREAK' statement can be written in the control block. This 'BREAK' statement forcibly terminates repetition control.
- A 'CONTINUE' statement can be written in the control block. This 'CONTINUE' statement causes control to branch to a statement that determines whether or not to repeat.

Rules for writing command

- A 'BREAK' statement can be written in any desired line in the control block.
- A 'CONTINUE' statement can be written in any desired line in the control block.
- Variables or constant values can be used in the initial and final values.
- A constant value can be used in the increment.

Example of source description

```
FOR [lab].W = 0 TO 10 STEP 1 ;lab is initialized to 0 which is repeated up to 10.
    :
NEXT
Expansion example
                MOV.W    #0,lab
            ..fr0000:
                CMP.W    #10,lab
                JEQ      ..fr0002
                    :
            ..fr0001:
                ADD.W    #1,lab
                JMP      ..fr0000
            ..fr0002:
```

FOR-NEXT Statement

Basic structure

- The basic structure of a FOR statement consists of structured description commands 'FOR' and 'NEXT' and a control block enclosed with these commands.

```
FOR Conditional expression
   Control block
NEXT
```

Function

- If the conditional expression is true, control branches to the immediately following control block.
- If the conditional expression is false, control branches to a line that immediately follows the structured description command 'NEXT.'
- System labels are generated for ‘FOR’ and ‘NEXT’.
- A 'BREAK' statement can be written in the control block. This 'BREAK' statement forcibly terminates repetition control.
- A 'CONTINUE' statement can be written in the control block. This 'CONTINUE' statement causes control to branch to the NEXT statement.
- A 'FOREVER' statement can be written in the conditional expression. The function of and rules for writing this statement are the same as for the FOR-STEP statement.

Rules for writing command

- Always be sure to enter a space or tab between FOR and the conditional expression.
- A 'BREAK' statement can be written in any desired line in the control block.
- A 'CONTINUE' statement can be written in any desired line in the control block.

Example of source description

```
FOR R0 <.S 10   ; Repeated as long as R0 is smaller than 10
   :
NEXT
Expansion example
                ..fr0000:
                    CMP.W    #10,R0
                    JGE      ..fr0002
                       :
                    JMP      ..fr0000
                ..fr0002
```

SWITCH Statement

Basic structure

- The basic structure of a SWITCH statement consists of structured description commands 'SWITCH,' 'ENDS,' and 'CASE' and a control block enclosed with CASE statement.

```
SWITCH expression
   CASE Data
            Control block
   CASE Data
            Control block
ENDS
```

Function

- System labels are generated for 'CASE' and 'ENDS'.
- Control branches to a control block immediately following the CASE command that holds data that matches the content of the expression written in the operand of the SWITCH statement.
- Evaluation is made on all CASE command data.

Rules for writing command

- Monomial and binomial expressions described in '6.4 Expressions' can be written in the operand expression of SWITCH.
- Be sure to write more than one instance of CASE statement. If no CASE is found between SWITCH and ENDS, the assembler outputs a warning.
- A constant can be written in the operand data of CASE.
- No value can be written in the operand data of CASE unless the value is fixed when assembled.
- No values can be written in the operand data of CASEs that are the same in one SWITCH statement.

BREAK

A 'BREAK' statement can be written at the end of a control block.

```
SWITCH expression
   CASE Data
            Control block
            BREAK
   CASE Data
            Control block
ENDS
```

Function

- The 'BREAK' statement causes control to branch to 'ENDS' unconditionally.

Rules for writing command

- The BREAK command must be written at the end of a control block.
- If this command is written in the middle of a control block, the assembler outputs a warning. In this case, although code for lines between the BREAK command and the next structured description command is generated, no code is generated for branching to that section.

DEFAULT

A structured description command 'DEFAULT' and a control block can be written at a position immediately preceding ENDS of a SWITCH statement.

```
SWITCH expression
   CASE Comparison data 1
            Control block
            BREAK
   CASE Comparison data 2
            Control block
   DEFAULT
            Control block
ENDS
```

Function

- If no matching data is found in the expression, control branches to the control block that immediately follows DEFAULT.
- A warning is output for CASE that is written between structured description command DEFAULT and ENDS. In this case, although object code for the control block immediately following this instance of CASE is generated, no code is generated for branching to that block.

Rules for writing command

- Only once instance of structured description command DEFAULT can be written in one SWITCH statement.

Example of source description

```
          SWITCH [ work ]
               CASE 1
                     :
                  BREAK
               CASE 2
                     :
               DEFAULT
                     :
          ENDS
Expansion example
   CMP.B    #1,work   ; Generated for CASE.
   JNE      ..sw0004  ; Generated for CASE.
     :
   JMP      ..sw0000  ; Generated for BREAK.
..sw0004:                       ; Generated for CASE.
   CMP.B    #2,work   ; Generated for CASE.
   JNE      ..sw0006  ; Generated for CASE.
     :
..sw0006:                       ; Generated for DEFAULT.
     :
..sw0000:                       ; Generated for ENDS.
```

DO Statement

Basic structure

- The basic structure of a DO statement consists of structured description commands 'DO' and 'WHILE' and a control block enclosed with these commands.

```
DO
   Control block
WHILE   Conditional expression
```

Function

- After executing the control block, the assembler checks the conditional expression written in the operand of WHILE to see if it is true or false.
- If the conditional expression is true, control branches to DO.
- If the conditional expression is false, control branches to the next line.
- A 'BREAK' statement can be written in the control block. This 'BREAK' statement causes control to branch to the line next to WHILE.
- A 'CONTINUE' statement can be written in the control block. This 'CONTINUE' statement causes control to branch to the WHILE statement.
- A 'FOREVER' statement can be written in the conditional expression. This statement causes control to branch to the DO statement unconditionally.
- Labels are generated for DO and WHILE.

Rules for writing command

- Always be sure to enter a space or tab between WHILE and the conditional expression.
- Expressions described in Conditional Expressions (Structured description Function) can be written in the conditional expression.

Example of source description

```
        DO
          :
        WHILE [lab].b ==1
```

Expansion example

```
        ..DO0000:
            :
          CMP.B   #1,lab
          JEQ     ..DO0000
        ..DO0002:
```

BREAK Statement

Function

- This statement generates an unconditional branch instruction.

Rules for writing command

- A BREAK statement can be written in the control block of FOR, DO, and SWITCH.
- A BREAK statement can be written in the control block of an IF statement providing that it exists in the control block of FOR, DO, or SWITCH statement.
- No BREAK statement can be written in the control block of an ordinary IF statement.

Example of source description

```
FOR [lab]=1 TO 10 STEP 1
    :
    BREAK
    :
NEXT
Expansion example
    MOV.W    #1,lab                 ; Generated for FOR.
..fr0000:                           ; Generated for FOR.
    CMP.W    #10,lab    ; Generated for FOR.
    JEQ      ..fr0002   ; Generated for FOR.
     :
    JMP      ..fr0002   ; Generated for BREAK.
     :
..fr0001:                           ; Generated for STEP.
    ADD.W    #1,lab                 ; Generated for STEP.
    JMP      ..fr0000   ; Generated for STEP.
..fr0002:                           ; Generated for NEXT.
```

CONTINUE Statement

Function

- This statement generates an unconditional branch instruction.

Rules for writing command

- A CONTINUE statement can be written in the control block of FOR and DO statements.
- A CONTINUE statement can be written in the control block of an IF or SWITCH statement providing that it exists in the control block of FOR or DO.
- No CONTINUE statement can be written in the control block of an ordinary IF or SWITCH statement.

Example of source description

```
FOR [lab]=1 TO 10 STEP 1
      :
   CONTINUE
      :
NEXT
Expansion example
   MOV.W    #1,lab                ; Generated for FOR.
..fr0000:                         ; Generated for FOR.
   CMP.W    #10,lab   ; Generated for FOR.
   JEQ      ..fr0002  ; Generated for FOR.
     :
   JMP      ..fr0001  ; Generated for CONTINUE.
     :
..fr0001:                         ; Generated for STEP.
   ADD.W    #1,lab                ; Generated for STEP.
   JMP      ..fr0000  ; Generated for STEP.
..fr0002:                         ; Generated for NEXT.
```

FOREVER Statement

Function

- This statement generates an unconditional branch instruction.

Rules for writing command

- A FOREVER statement can be written in the conditional expression of FOR and DO-WHILE statement.
- A conditional expression consisting of FOREVER is always true.
  FOR FOREVER
  WHILE FOREVER

Example of source description

```
FOR     FOREVER
   :
NEXT
Expansion example
..fr0000:
   :
   JMP      ..sfr0000
..fr0002:
```

Assignment Statement

Basic structure

- An assignment statement consists of a substitute command (=) and the left and right sides of the statement.

Function

- An assignment statement substitutes the calculation result of the expression on the right side of the statement for a variable on the left side. There are following types of assignment statements:

Operations	content
=	Substitutes an unsigned value for the left side.
=.S	Substitutes a sign-extended value on the right side for the left side.
=.Z	Substitutes a zero-extended value on the right side for the left side.
=.EL	Generates a LDE command.
=.ES	Generates a STE command.

Rules for writing assignment statement

- No expressions that contain unary or binary operators can be written on the right side of assignment statement '=.S,' '=.Z,' '=.EL,' or '=.ES.'
- Variables listed below can be written on the left and right sides of assignment statements '=.S' and '=.Z':
  - Memory variables (except for [SP] relative)
  - Data register and address register indirect among register variables
- The variables that can be written on the left and right sides of assignment statement '=.EL' are those whose contents can be written in the operands 'dest' and 'src' of mnemonic 'LDE.'
- The variables that can be written on the left and right sides of assignment statement '=.ES' are those whose contents can be written in the operands 'dest' and 'src' of mnemonic 'STE.'

Precautions

For details about mnemonics, refer to the "M16C Family Software Manual."

- A warning is output if an entirely same variable is written on the left and right sides of an assignment statement.
- If a different type of variable is substituted for, no expressions can be written on the right side of the assignment statement that contains unary or binary operators.

Combination of variable types that can be written in assignment statement (=)

Left side(Type)	Right side(Type)			
	Byte	Word	Address	Long word
Byte	Enable	Disable	Disable	Disable
Word	Disable	Enable	Disable	Disable
Address	Disable	Disable	Enable	Disable
Long word	Disable	Disable	Disable	Enable

Combination of variable types that can be written in sign-extended assignment statement (=.S)

Left side(Type)	Right side(Type)			
	Byte	Word	Address	Long word
Byte	Disable	Disable	Disable	Disable
Word	Enable	Disable	Disable	Disable
Address	Disable	Disable	Disable	Disable
Long word	Disable	Enable	Disable	Disable

Precautions
If for a 'word type =.S byte type' assignment expression, "R2" or "R3" is specified for the left side of the expression, the assembler uses the "R0" register.
If for a 'long word type =.S word type' assignment expression, "memory variable" or "R3R1" is specified for the left side of the expression, the assembler uses the "R2R0" register pair.

Combination of variable types that can be written in zero-extended assignment statement (=.Z)

Left side(Type)	Right side(Type)			
	Byte	Word	Address	Long word
Byte	Disable	Disable	Disable	Disable
Word	Enable	Disable	Disable	Disable
Address	Enable	Enable	Disable	Disable
Long word	Enable	Enable	Enable	Disable

Precautions
If for a 'word type =.Z byte type' assignment expression, "R2, "R3" is specified for the right side of the expression, the assembler uses the "R0" register.

Combination of variable types that can be written in special assignment statements (=.EL, =.ES)

Left side(Type)	Right side(Type)			
	Byte	Word	Address	Long word
Byte	Enable	Disable	Disable	Disable
Word	Disable	Enable	Disable	Disable
Address	Disable	Disable	Disable	Disable
Long word	Disable	Disable	Disable	Disable

Description example of assignment statement and its expansion example

Example of source description	Expansion example
R1 = R0	MOV.W R0,R1
R0 = R0 + 2	ADD.W #2,R0
R0 =.S R0L	EXTS.B R0L
R0 =.Z R0L	MOV.B #0,R0H
R0L =.EL [lab].B	LDE.B lab,R0L
[lab].W =.ES R0	STE.W R0,lab
R0 =.S R0L	EXTS.B R0L
R0 =.S R0H	MOV.B R0H,R0L EXTS.B R0L
[lab_w].W =.S R0L	MOV.B R0L,lab_w EXTS.B lab_w
R2R0 =.S R0	EXTS.W R0
R2R0 =.S R1	MOV.W R1,R0 EXTS.W R0
[lab_l].L =.S R0	EXTS.W R0 MOV.W R0,lab_l MOV.W R2,lab_l+2
R0 =.Z R0L	MOV.B #0,R0H
R0 =.Z R0H	MOV.B R0H.R0L MOV.B #0,R0H
[lab_w].W =.Z R0L	MOV.B R0H,lab_w MOV.B #0,lab_w+1
[lab_a].A =.Z R0	MOV.W R0L,lab_a MOV.B #0,lab_a+2
R0L =.EL [lab_b]	LDE.B lab_b,R0L
[lab_w].W =.ES R0	STE.W R0.lab_w

Structure of Structured Description Commands

This section shows structured description statements that can be written in AS30 programming. When writing structured description, please follow the syntax shown below.

Definition of Terms

The following explains the description terms used in this section. The variable name or operator indicated by each term can be written at the position where the term is written.

Register variable

Term	Contents
regb	R0L,R0H,R1L,R1H,A0.B,A1.B,[A0.B],[A1.B]
regw	R0,R1,R2,R3,A0,A1,[A0],[A1]
regc	FB,SB,SP,ISP,FLG,INTBH,INTBL
reglw	R2R0, R3R1
regad	A1A0

Precautions
SP refers to the stack pointer (user stack pointer or interrupt stack pointer) indicated by the U flag. For details about the stack pointer and U flag functions, refer to the "M16C Family Software Manual."

Special register variable

Term	Contents
dsp:8[SP]	Special Page addressing variable
INTB	INTB
IPL	IPL
[STK]	[STK]

Precautions
Memory variable except for bit variable can be written for "dsp".

Memory variable

Term	Contents
memb	Byte type memory variable (except for description of "SP")
memw	Word type memory variable (except for description of "SP")
mema	Address type memory variable
meml	Long word type memory variable
regmembit	Register bit variable, memory bit variable
flgbit	Flag variable

Operators

Term	Contents
Unary operators	~, -, ++, --
Binary operators 1	+, +.C, -, -.C
Binary operators 2	+.C, +.CD, -.C, -.CD
Binary operators 3	*, *.S
Binary operators 4	/, /.S, %, %.S, %.SE
Binary operators 5	&, |, ^, ?
Binary operators 6	>>.C, <<.C
Binary operators 7	<>.R
Binary operators 8	<>.A, <>.L
Relational operators	==, !=, >, >.S, <, <.S, =>, =>.S, <=, <=.S
Coincidence comparing operators	==, !=
Logical operators	&&, ||
Constants	Numeric value or expression value that is fixed when assembled

Syntax of Statements

The following shows the syntax of statements.

Uo = Unary operator
Bo = Binary Operator
Ro = Relational operator
Co = Coincidence comparing operator
Lo = Logical operator

Simple assignment statements and assignment statements containing unary operators

Left side is Memory variable

```
memb    =     <constant>
memb    =     <Uo> memb
memb    =     <Uo> regb
memb    =     [STK].B
memb    =.ES  memb,regb
memw    =     <constant>
memw    =.S   <Uo> memw
memw    =.S   <Uo> regw
memw    =.S   memb
memw    =.S   regb
memw    =.Z   memb
memw    =.Z   reg
memw    =     [STK].W
memw    =.ES  memw,regw
mema    =     <constant>
mema    =     mema
mema    =.Z   memb
mema    =.Z   memw
mema    =.Z   regb
mema    =.Z   regw
memlw   =     <constant>
memlw   =     meml
memlw   =     R2R0
memlw   =     R3R1
memlw   =     A1A0
memlw   =.S   memw
memlw   =.S   regw
memlw   =.Z   memb
memlw   =.Z   memw
memlw   =.Z   mema
memlw   =.Z    regb
memlw    =.Z  regw
```

Precautions
Only the data register variables can be written for "regb" and "regw" in "=.S" and "=.Z."

Left side is Register

```
regb        =       <constant>
regb        =       <Uo> memb
regb        =       <Uo> regb
regb(Except for A0.B and A1.B)= [STK].B
regw        =       <constant>
regw        =       <Uo> memw
regw        =       <Uo> regw
regw        =.S     memb
regw        =.S     regb
regw        =.Z     memb
regw        =.Z     regb
regw        =       [STK].W
regl        =       <constant>
regl        =       meml
regl        =       R2R0
regl        =       R3R1
regl        =       A1A0
regl        =.S     memw
regl        =.S     regw
regl        =.Z     memb
regl        =.Z     memw
regl        =.Z     mema
regl        =.Z     regb
regl        =.Z     regw
regc        =       <constant>
regc        =       memw
regc        =       regw
regc        =       [STK].W
R0,R1,R2,R3, A0,A1,SB,FB=[STK].W(Multiple register can be written in left side)
```

Left side is Register or Memory variable

```
memb, regb  =.EL    memb
memw, regw =.EL     memw
memw, regw =        regc
memb,regb   =       dsp:8[SP]
memw,regw   =       dsp:8[SP]
mema, [A0.A], [A1.A], R2R0, R3R1, A1A0      =       regpc
```

Left side is Special Register

```
INTB        =       <constant>
IPL                 =       <constant>
dsp:8[SP]   =       memb,regb
dsp:8[SP]   =       memw,regw
[STK].B     =       <constant>
[STK].B     =       memb
[STK].B     =       regb (Except for A0.B and A1.B)
[STK].W     =       <constant>
[STK].W     =       memw
[STK].W     =       regw
[STK].W     =       regc
[STK].W     = R0,R1,R2,R3,A0,A1,SB,FB (Multiple register can be written)
[STK].A     =       mema
```

Left side is bit variable

```
regmembit = 1, 0, ~regmembit(Bit name is same as left side)
flgbit      =       1, 0
```

Assignment statements containing unary operators

```
memb/regb  =        Uo memb/regb
memw/regw  =        Uo memw/regw
```

Assignment statements containing binary operators 1

```
memb/regb  =        [Uo] memb/regb Bo 1 constant/memb/regb
memw/regw  =        [Uo] memw/regw Bo 1 constant/memw/regw
```

Assignment statements containing binary operators 2

```
memb/regb  =        [Uo] memb/regb Bo 2 constant/memb/regb
memw/regw  =        [Uo] memw/regw Bo 2 constant/memw/regw
```

Assignment statements containing binary operators 3

```
memw/regw  =        [Uo] memb/regb Bo 3 constant/memb/regb
meml/regl  =        [Uo] memw/regw Bo 3 constant/memw/regw
```

Assignment statements containing binary operators 4

```
memb/regb  =        [Uo] memb/regb Bo 4 constant/memb/regb
memw/regw  =        meml/reglw/regad Bo 4 constant/memw/regw
```

Assignment statements containing binary operators 5

```
memb/regb  =        [Uo] memb/regb Bo 5 constant/memb/regb
memw/regw  =        [Uo] memw/regw Bo 5 constant/memw/regw
```

Assignment statements containing binary operators 6

```
memb/regb  =        [Uo] memb/regb Bo 6 constant
memw/regw  =        [Uo] memw/regw Bo 6 constant
```

Assignment statements containing binary operators 7

```
memb/regb  =        [Uo] memb/regb Bo 7 constant/R1H
memw/regw  =        [Uo] memw/regw Bo 7 constant/R1H
```

Assignment statements containing binary operators 8

```
memb/regb  =        [Uo] memb/regb Bo 8 constant/R1H
memw/regw  =        [Uo] memw/regw Bo 8 constant/R1H
meml/reglw/regad    = meml/reglw/regad Bo 8 constant/R1H
```

Syntax of expression 1

[Uo] memb/regb
[Uo] memw/regw

```
Expression 2
Expression 2 Ro        Immediate/memb/regb
Expression 2 Ro        Immediate/memw/regw
Expression 2 Lo        Expression 2
Expression 3 Lo        Expression 3
Expression 3
regmembit/flgbit
```

Syntax of expression 2

Among syntaxes indicated on the right side of the assignment expression, all syntaxes except for the following contents can be written.

- Registers and stacks listed below

```
FB, SB, SP, ISP, FLG, INTBH, INTBL, INTB, IPL and [STK]
```

- Expressions where multiplication results in 32 bits
- Inverted expressions of register bit and memory bit variables

```
~regmembit
```

Syntax of expression 3

```
Binomial expression .b        Ro        Constant/memb/regb
Binomial expression .w        Ro        Constant/memw/regw
regmembit/flgbit        = coincidence comparing operator 1/0
```

Syntax of Conditional Expression

IF statement

```
IF Expression 1
```

FOR-STEP statement

```
FOR        variable= [Uo]variable/constant TO variable/constant STEP constant
```

FOR-NEXT statement

```
FOR        Expression 1
```

WHILE statement

```
WHILE        Expression 1
```

SWITCH statement

```
SWITCH        Expression
```

M16C,R8C Family C Compiler Package V.5.45
Assembler User's Manual

Publication Date: Apr. 1, 2010 Rev.2.00

Published by: Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi,
Kanagawa 211-8668 Japan

Edited by: Renesas Solutions Corp.

M16C Series,R8C Family
C Compiler Package V.5.45
Assembler User's Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

RE10J2006-0200