NEC

User's Manual

AS6133 Ver.2.21 or Later

Assembler

For PC-9800 (MS-DOS™ Based)

For IBM PC/AT™ (PC DOS™ Based)

Target Devices
μPD6133 Series
μPD6604 Series
μPD63 Series
μPD67 Series

Document No. U10115EJ3V0UM00 (3rd edition)
Date Published August 2001 J CP(K)

[MEMO]

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-3067-5800
Fax: 01-3067-5899

NEC Electronics (France) S.A.
Madrid Office
Madrid, Spain
Tel: 091-504-2787
Fax: 091-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Guarulhos-SP, Brasil
Tel: 11-6462-6810
Fax: 11-6462-6829

J01.2

Major Revisions in This Edition

Page	Description
Throughout	Deleting description "separate volume of SM6133" or "supplied with SM6133" because AS6133 assembler is separated from SM6133 simulator
	Changing supported debugger from NEC's SM6133 simulator to Naito Densei Machida Mfg's EB-6133
	Adding target device
PREFACE	Changing description of **2.1 PC-9800 Series**, **2.2 IBM PC/AT Compatibles**, and **3.1 Assembler**
p.45	Adding description to **PART I**, **3.4 Pseudo Instructions and Control Instructions**
p.74	Adding **PART II**, **1.2 Supported Debugger**
p.75	Adding **2.2 Install**
p.77	Adding **Table 3-1 Device Name That Can Be Described and Supported Device**
p.87	Changing description of **[Example]** in **4.4.2 Starting the assembler**
p.102	Adding description to **5.4.1 Error check for instructions exceeding the allowable number of bits**
p.103	Changing description in **5.4.3 Check for the destination of a branch instruction (automatic check on BANK0 and BANK1)**
p.104	Adding description to **5.4.5 Check for input/output instructions for nonexisting ports**
p.112	Adding **APPENDIX A CONSTRAINTS**
p.113	Adding **APPENDIX B REVISION HISTORY**

The mark ★ shows major revised points.

PREFACE

★ 1. The AS6133 assembler supports under the following 4-bit microcontrollers.

Series Name	Supported Device
μPD6133	μPD6132, 6132A, 6133, 6134, 6135, 61P34B
μPD6604	μPD6603, 6604, 6605, 66P04B
μPD63	μPD62, 62A, 63, 63A, 64, 64A, 6P4B, 65, 6P5
μPD67	μPD67, 68, 69, 6P9

2. The AS6133 assembler runs under the following environment:

★ 2.1 PC-9800 Series

(1) Supported PC-9800 series OS

MS-DOS Ver.5.0 or later[Note 1]
Windows™ 3.1/95/98[Note 2]
WindowsNT™ 4.0[Note 2]

Notes 1. Versions 5.00 and 5.00A feature a task swap function. However, this software does not support the use of this function.
2. Can be used with MS-DOS prompt (Windows 3.1/95/98) or command prompt (WindowsNT).

The AS6133 assembler runs on MS-DOS for NEC's PC-9800 series or Windows, or Windows for PC-9800 series supplied by Microsoft.

NEC will not be liable for unsatisfactory operation of this assembler under another commercially available version of MS-DOS or Windows.

The CONFIG.SYS file must contain the following settings:

- files = 15 (15 or more)
- buffers = 10 (10 or more)

★ ### 2.2 IBM PC/AT Compatibles

(1) Supported IBM PC/AT compatibles

IBM PC/AT compatible personal computers on which the following OS runs:

(2) Supported IBM PC/AT compatible OSs

MS-DOS Ver.6.0 or later[Note 1]
PC DOS Ver.6.1 or later[Note 1]
Windows 3.1/95/98[Note 2]
WindowsNT 4.0[Note 2]

Notes 1. Versions 6.0 and 6.1 feature a task swap function. However, this software does not support the use of this function.

2. Can be used with MS-DOS prompt (Windows 3.1/95/98) or command prompt (WindowsNT).

The AS6133 assembler runs on MS-DOS or Windows for IBM PC/AT compatibles supplied by Microsoft, or PC DOS for IBM PC/AT compatibles supplied by IBM Japan.

NEC will not be liable for unsatisfactory operation of this assembler under another commercially available version of MS-DOS, Windows, or PC DOS.

★ ## 3. Supply Media

3.1 Assembler

(1) File name

AS6133.EXE

(2) Floppy disk types

PC-9800 series: High-density 3.5-inch floppy disk (3.5" 2HD)
IBM PC/AT compatibles: High-density 3.5-inch floppy disk (3.5" 2HD)

4. Symbols Used in This Manual

...	The preceding option may be repeated any number of times.
[ ]	The options enclosed in parentheses may be omitted.
{ }	Only one of the options in the braces must be selected.
Δ	One single-byte space or TAB.
" "	Used to enclose a character or character string.
CR	Carriage return
LF	Line feed
TAB	Horizontal tab
○○○	Represents any character string.
×××	Represents any character string.
□□□	Represents any character string.
≡	Indicates corresponding contents.
< >	Represents data equivalent to the enclosed item.

5. File Naming Rules

```
[drive-name:]   [\directory-name\...] file-name   [.extension]
```

drive-name:	Drive in which the floppy disk containing the file is mounted. Omitting the drive name causes the current drive to be assumed.
file-name:	String of up to eight single-byte or four double-byte characters.
extension:	String of up to three single-byte characters.

CONTENTS

PART I LANGUAGE

LIST OF FIGURES

LIST OF TABLES

PART I

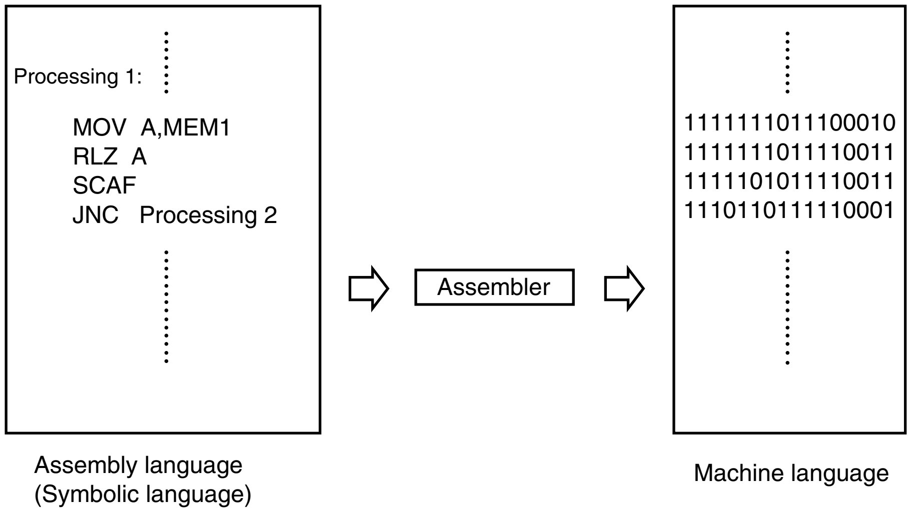
LANGUAGE

CHAPTER 1 OVERVIEW

1.1 Overview of the Assembler

1.1.1 What is an assembler?

A microcontroller can only interpret its so-called machine language, which consists entirely of 0s and 1s. Machine language is very complicated for humans to understand and essentially impossible to remember. By assigning symbolic (assembly) language instructions to machine language instructions, however, programs can be coded in such a way that humans can more easily understand them. An assembler is a program which translates this "human-friendly" symbolic language, into the machine language of the microcontroller.



Assemblers can be classified as absolute assemblers or relocatable assemblers.

AS6133 is an absolute assembler. Unlike conventional absolute assemblers, however, it allows split programming. Thus, although it is actually an absolute assembler, AS6133 can be said to have characteristics similar to those of a relocatable assembler.

1.1.2 What is an absolute assembler?

A machine language instruction consists of an instruction and data. An instruction specifies an operation to be performed by the microcontroller. Data is the value(s) on which that operation is performed.

Data can include the constants and variables to be used to perform an arithmetic instruction.

An absolute assembler makes the addresses assigned to instructions and data absolute upon translating them into machine language. This means that addresses and data must be determined before the program is assembled. The information is passed to the assembler by the location counter control pseudo instruction called "ORG".

The machine language code created by the absolute assembler is stored in memory as is and executed by the microcontroller. The machine language code thus created is called an absolute object module. On the other hand, the source symbolic language code is called a source module.

1.1.3 What is a relocatable assembler?

The absolute object module created by an absolute assembler has absolute data and addresses. On the other hand, an assembler which creates an object module which can be relocated to any address in memory is called a relocatable assembler. The machine language code created by a relocatable assembler is called a relocatable object module.

A relocatable object module cannot be directly executed as a program by the microcontroller. This is because addresses and data are relative. A program which translates a relocatable object module such that it can be executed by the microcontroller is called a linker.

What is a linker?

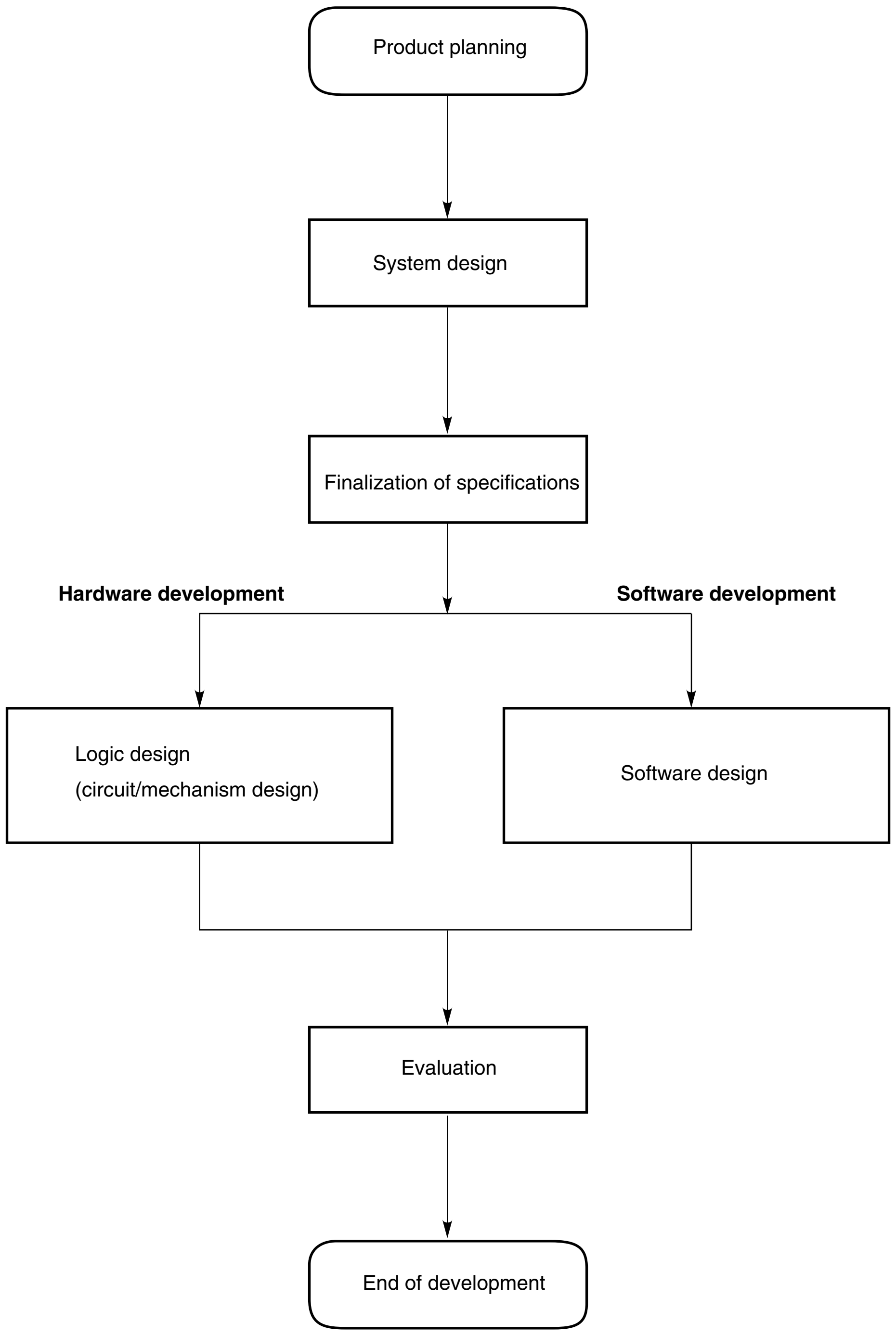
A linker determines the location of one or more relocatable object modules created by a relocatable assembler, resolves address references, and combines the modules into one. It also assigns absolute values to those addresses and data to which relative values were assigned.

The combination of modules produced by the linker is called a load module. This load module cannot be directly executed by the microcontroller. It must, therefore, be translated into a form that can be executed by the microcontroller.

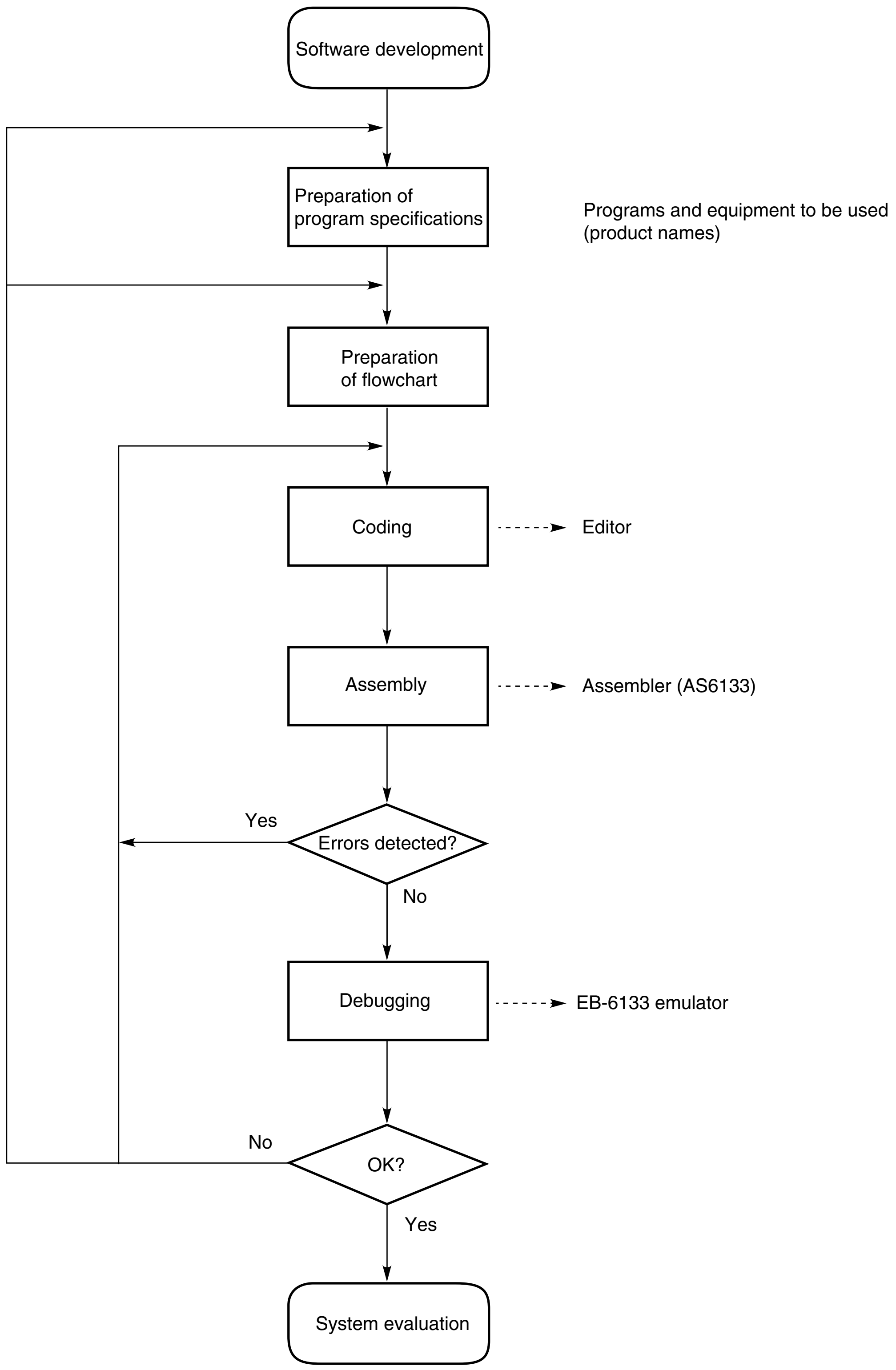
1.1.4 Flow of system development using the μPD6133 Series

Figure 1-1 shows the flow of system development using the μPD6133 Series. Figure 1-2 shows the flow of software development in detail.

Figure 1-1. System Development Flow



★ **Figure 1-2. Software Development Flow**



1.1.5 Comparison of assemblers

Table 1-1 lists the features of absolute and relocatable assemblers.

Table 1-1. Comparison of Assemblers

		Absolute Assembler	Relocatable Assembler
Assembly method		Batch assembly (AS6133 allows pseudo split assembly.)	Split assembly ↓ Link
Addresses on the assembly list		Absolute	Relative
Variable	Restrictions on operations performed on variables in the operand part	None	Imposed by the linker.
	Local variable	Cannot be defined. (Can be defined for AS6133.)	Can be defined.
Others		Because batch assembly is used, partially modifying the source does not reduce the assembly time. (AS6133 can realize a reduction, however, because it allows pseudo split assembly.)	Address calculation is necessary during debugging. Because split assembly is supported, module-by-module programming by more than one person is possible.

1.2 Functional Overview of μPD6133 Series Assembler

1.2.1 Creating a sequence file

The μPD6133 series assembler (AS6133) is an absolute assembler. Despite being an absolute assembler, AS6133 supports module programming, one of the features of a relocatable assembler. Unlike relocatable assembler packages, however, AS6133 does not provide a linker program. The features of a linker are, however, supported.

When programming source modules, a sequence file which describes the order in which the source module files are to be linked is necessary. The sequence file also specifies device names and assembly-time options, in addition to the order in which the source module files are to be linked.

1.2.2 Creating source module files

When designing a program, it is generally divided into several subprograms, according to function. If the functional independence of the subprograms is high, the debugging of each subprogram will be easy. This enhances the development efficiency and will lead to better maintainability in the future.

A subprogram is a unit of coding and also acts as the unit of input to the assembler. The unit of input to the assembler is called a source module.

Once the coding of a source module has been completed, use an editor to write the module to a file. The created file is called a source module file.

When a source program is split into source modules, the order in which the source modules are to be linked must be written in a sequence file.

Splitting into files using INCLUDE statements differs from the above-mentioned splitting into source modules. More specifically, a file specified by an INCLUDE pseudo instruction can be said to be part of the source module containing the INCLUDE pseudo instruction.

1.2.3 Supported Japanese code

AS6133 can assemble source programs written in Japanese code (8-bit JIS code and shift JIS code).

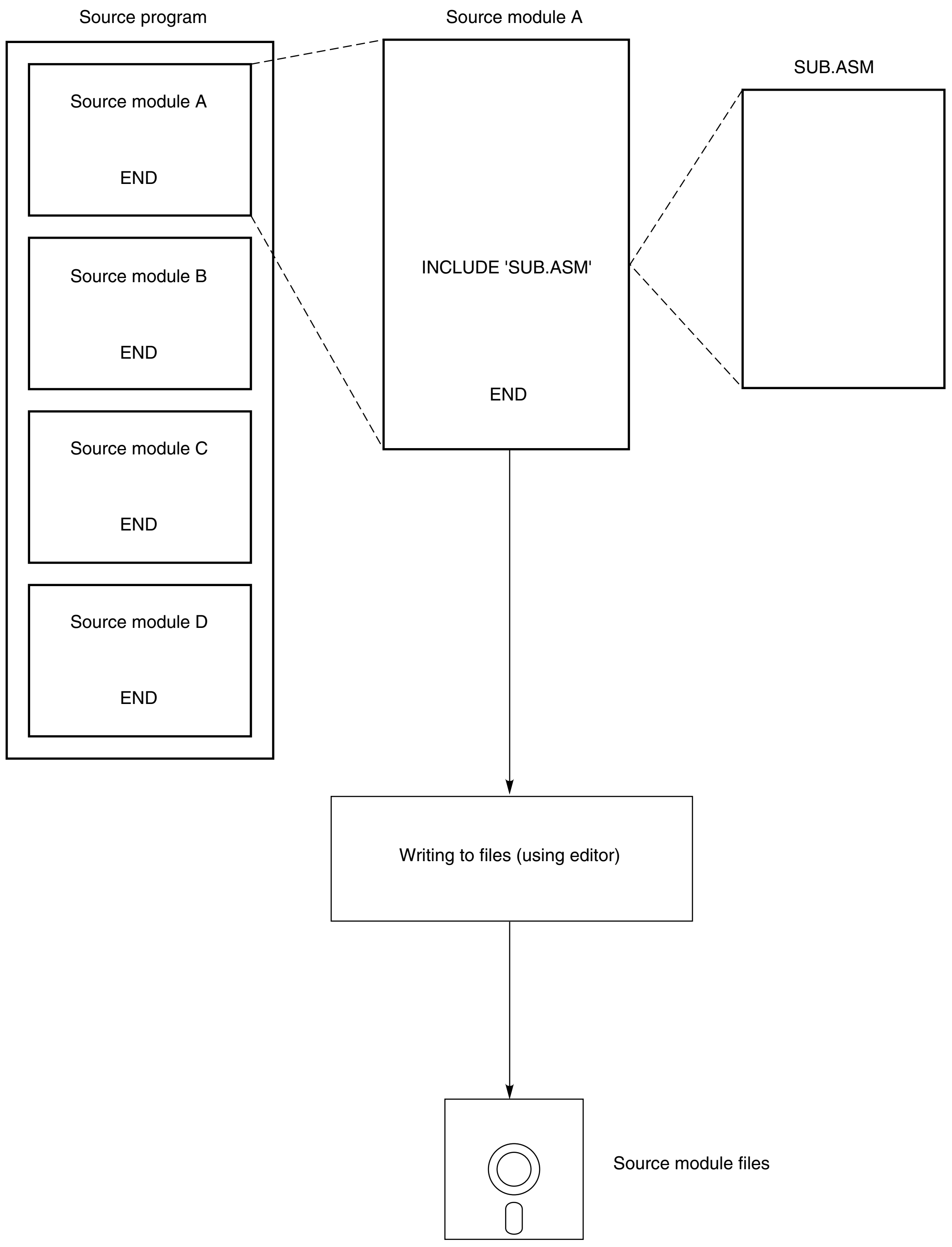
Japanese code can be used not only in comment fields but also in symbol fields.

1.2.4 External module definition reference function

The PUBLIC and EXTRN pseudo instructions can be used to reference symbols defined in external modules. A symbol for which PUBLIC is declared can be referenced with the EXTRN declaration at any time.

The symbols defined in backward modules can be referenced at assembly time while those defined in forward modules can be referenced at link time.

Figure 1-3. Creating Source Module Files



1.2.5 Assembly

To assemble a source module, the following files are necessary:

- Assembler (AS6133.EXE)
- Source module file (○○○○.ASM, ××××.ASM, etc.)
- Sequence file (□□□□.SEQ)

When starting AS6133, the output list can be controlled directly from the console or by specifying assembly options in the sequence file.

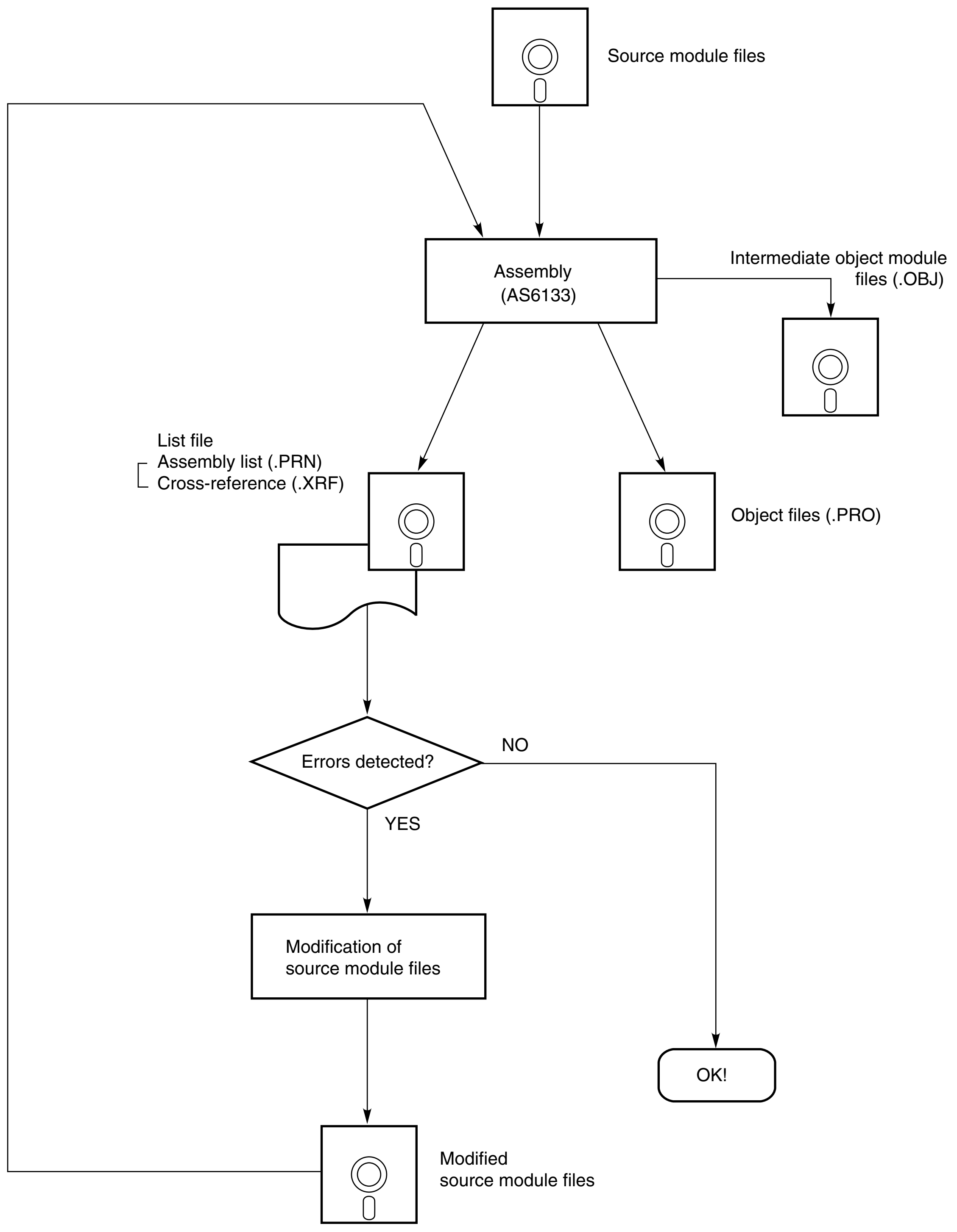
If any errors are found in the assembly list, modify the source modules and repeat assembly until all errors have been removed.

If the source program consists of modules, AS6133 creates intermediate object module files (.OBJ) at assembly. These intermediate object module files are used when the source program is partially modified and re-assembled.

To reduce the assembly time, AS6133 re-assembles modified source modules only, using the already created intermediate object module files for those source modules that have not been modified. To check whether a source module has been modified, the assembler compares the creation date and time of the source module file with that of the intermediate object module file having the same name. If the source module file is found to be newer, it is judged to have been modified. If, therefore, the intermediate object module file corresponding to a source module file is not found, or if the source module file is older, the assembler automatically detects this and creates an intermediate object module file at assembly.

The assembly time reduction function can reduce the assembly time considerably as the user proceeds with debugging.

Figure 1-4. Creating Object Files



1.3 Before Starting Program Development

This section explains those items with which the user must be familiar to enable efficient use of AS6133.
The subsequent sections provide a detailed explanation of the above.

1.3.1 Restrictions on symbols

(1) Restrictions on the number of symbols

Each source module can use a symbol table area of up to 64 KB.
With AS6133, one symbol can be defined using up to 253 characters (one byte per character).[Note]
The maximum number of symbols that can be used is as follows:
240 if all symbols are 253 characters long.
3,368 if all symbols are eight characters long.

Note Double-byte characters (shift JIS codes) consist of two bytes each.

(2) Symbols in a macro

Those symbols which are not declared as being global are handled as local symbols.

1.3.2 Restrictions on pseudo instructions

The MACRO, REPT, and IF statements can be nested to up to 40 levels deep. When expanding pseudo instructions in a pseudo instruction, care must be exercised to prevent the nesting level from exceeding 40.
In a macro, macro name references can be made but macro definitions cannot be created.

Table 1-2. Pseudo Instructions Which Can Be Nested and Maximum Nesting Levels

Pseudo Instruction Which Can Be Nested	Maximum Nesting Levels	Total
REPT-EXITR-ENDR	8	40
IF-ELSE-ENDIF	40	
MACRO-ENDM	40	
INCLUDE	8	8[Note]

Note The INCLUDE statement can be nested independently of the above pseudo instructions.

1.3.3 Notes on using Japanese code

A source list can be created by using a Japanese editor.
The available character codes are 8-bit JIS (single-byte) codes and shift JIS (double-byte) codes.
Reserved words must be written using single-byte codes.
A double-byte space, colon, and semicolon must not be used to delimit symbol, mnemonic, and operand fields. The single-byte and double-byte codes for characters are different.
For example, when a space is coded using a double-byte code, it is handled as a blank, not as a delimiter.

1.3.4 Setting the date and time of the host machine

Always check the current date and time when starting MS-DOS on the host machine (PC-9800 series).

At assembly time, AS6133 compares the creation dates and times of the source module files with those of the intermediate object module files having the same names. As a result of this comparison, if an intermediate object module file is found to be newer, the corresponding source module is not assembled.

If the time indicated by the clock of the host machine is subsequent to the creation date and time of a source module file, changes made to the source module file may not take effect after assembly.

1.3.5 Restrictions on the number of source modules

AS6133 can assemble a source program consisting of up to 30 modules.

The source modules are handled as a single source program by describing the assembly order in a sequence file (.SEQ).

CHAPTER 2 CODING SOURCE PROGRAMS

2.1 Basic Configuration of a Source Program

A source program consists of one or more source modules, as shown in Figure 1-3. Each source module consists of one or more statements. The configuration of a statement is shown in **Section 2.2**.

No restrictions are imposed on the size of a source module. This means that any number of statements can be written. A source program can consist of up to 30 source modules.

In a source program, instructions, pseudo instructions, and control instructions can be written at any location. The END pseudo instruction, however, can be written only at the end of each source module.

The END pseudo instruction need not be written in an include file to be read into a source module by the INCLUDE pseudo instruction in the source module.

2.2 Configuration of a Statement

A source program in assembler language consists of statements.

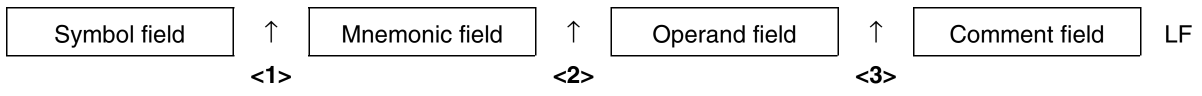
A statement is written using the characters listed in **Section 2.4**.

When creating a source program using a text editor, each statement is terminated with a CR (carriage return) code and an LF (line feed) code. The assembler regards an LF code as being a statement terminator, but ignores a CR code.

A statement consists of four fields: symbol, mnemonic, operand, and comment, as shown below.

Each field must be delimited with a single-byte space ( ) (8-bit JIS code 20H), TAB code (09H), single-byte colon (:) (3AH), or single-byte semicolon (;) (3BH). Up to 256 characters can be written on one line.

The format of a statement is arbitrary. The symbol, mnemonic, operand, and comment fields can start in any columns provided that they appear in this order. A statement containing only a symbol or comment field, as well as an empty statement can also be written.



<1> To enter a symbol in a symbol field, use a single-byte colon or blank (one or more single-byte spaces or TAB code) as a delimiter.
Whether a colon or blank should be used depends on the instruction to be written in the mnemonic field.

<2> When an operand field is necessary, use a blank as a delimiter.

<3> When a comment is written in the comment field, use a single-byte semicolon as a delimiter.

<4> Any number of blanks can be inserted before and after a colon or semicolon.
In Example 1, a colon is used to delimit the symbol and mnemonic fields. In Example 2, a blank is used.

[Example 1]

```
AAA  :  MOV     A,#8H     ;  A=8H
```

[Example 2]

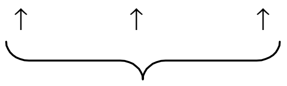
```
AAA  EQU  7
```

2.3 Tabulation Function

AS6133 provides a tabulation function to improve the readability of an assembly listing. The tabulation function re-arranges the symbol, mnemonic, operand, and comment fields in a source program so that they each begin in a column that is a multiple of eight.

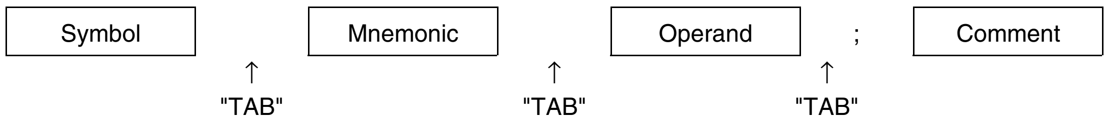
[Example] Addition:

```
MOV     A,#8H
MOV     R01,A    ;R01=8H
```



Columns that are multiples of eight (column numbers equal to the tab number, multiplied by eight)

To use the tabulation function, insert a TAB (Horizontal TAB, 09H) code in the source program before each of the mnemonic and operand fields and before the single-byte semicolon (;) indicating the start of a comment field.



AS6133 supports an assembly option which allows the user to select whether the TAB code (09H) should be sent to the printer or replaced by single-byte spaces, depending on the printer being used.

This option is provided to support printers which cannot recognize TAB codes.

In this way, AS6133 allows the user to specify that a TAB code should be converted to single-byte spaces before being sent to the printer.

Remark It is recommended that the TAB code be used to make effective use of the disk.

2.4 Character Sets

The 8-bit JIS code set and the shift JIS code set must be used to write statements.

Restrictions are imposed on the characters that can be used for symbols. For details, see **Section 2.5.1**. Reserved words can be used either single-byte alphabetic upper or lower cases. Symbols defined by the user, however, are case sensitive.

[Example 1]

```
AAA     EQU  3
Aaa     EQU  5
```

AAA and AAa are regarded as being different symbols.

[Example 2]

```
MOV     MEM1,#1
Mov     mem1,#3
```

MEM1 and mem1 are different symbols. MEM1 is set to 1 and mem1 to 3.
The reserved word MOV, however, is interpreted as being identical to Mov.

2.4.1 Alphanumeric characters

Single-byte alphabetic characters and arabic numerals are collectively referred to as alphanumeric characters.

2.4.2 Digits

Binary digits:	The two digits 0 and 1 are referred to as binary digits.
Octal digits:	The eight digits 0, 1, 2, 3, 4, 5, 6, and 7 are referred to as octal digits.
Decimal digits:	The ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 are referred to as decimal digits.
Hexadecimal digits:	The sixteen digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F are referred to as hexadecimal digits.

2.4.3 Special characters

The following special characters are single-byte characters. The equivalent double-byte characters cannot be used as special characters; they are interpreted as is.

These special characters (except the LF code) can be used to represent their normal meanings in character strings (character constants) and in comment fields.

Single-byte character	Name	Main use
	Space	Field delimiter
?	Question mark	Equivalent to an alphabetic character.
@	Unit price symbol	Specifies indirect addressing.
_	Underscore	Equivalent to an alphabetic character.
,	Comma	Operand delimiter
.	Period	Bit segment operator
+	Plus	Plus sign or addition operator
–	Minus	Minus sign or subtraction operator
*	Asterisk	Multiplication operator
/	Slash	Division operator
(	Opening parenthesis	Change operation order.
)	Closing parenthesis	
$	Dollar sign	Value of the location counter
=	Equal sign	Comparison operator
;	Semicolon	Indicates the start of a comment.
;;	Double semicolon	Indicates the start of a comment in a macro.
:	Colon	Label delimiter
'	Quotation mark	Indicates the start or end of a character constant.
<		Comparison operators
>		
#		Specifies immediate data.
&	Ampersand	Specifies the concatenation of character strings in a macro.
%	Expression operator	Used immediately before a macro parameter to indicate the transfer of a value.
TAB code	Horizontal tab	Equivalent to eight spaces.
LF code	Line feed	Statement terminator
CR code	Carriage return	Normally ignored by the assembler.

2.5 Symbol Field

When data or an address to be used in an instruction or pseudo instruction is written using a numeric value or numeric expression, AS6133 recognizes this as being an error. To use data or an address, use a name which enables its easy recognition. The name assigned to data or an address is called a symbol.

A symbol can be entered in a symbol field. This is referred to as defining a symbol.

A symbol to be used in a program can appear anywhere in the program provided that it is declared before use. The scope of a symbol depends on where it is declared.

Symbols having the same name cannot be used in a module. They can, however, be used in different modules because symbols are basically used locally within a module. To use a symbol globally in more than one module, the symbol must be declared as being public.

Symbols are classified as labels and names according to their purpose and how they are defined.

(1) Names

The symbols defined in the EQU and SET pseudo instructions are called names. Numeric data or addresses can be assigned to names. These names can be used in a program instead of numeric data and addresses. Thus, numeric data can be used indirectly by assigning it a name.

[Example]

```
DATA1     EQU    8H
```

The name DATA1 is defined for numeric data 8H.

(2) Labels

Labels are symbols which can be assigned to the address of an instruction (mnemonic) or to the ORG, DW, or DT pseudo instruction. A label is used to reference the program memory address (value of the location counter) assigned to the instruction or pseudo instruction to which the label is assigned.

Thus, a label can be written at the top address of a routine, with a name indicating the processing of the routine, thus causing a branch from another routine to the routine or to reference the routine.

[Example]

```
LOOP:MOV      A,@R0H
      .
      .
      .
     JMP      LOOP
```

In this example, LOOP is a label.

2.5.1 Rules governing the writing of symbols

The following rules are applied to writing symbols:

(1) Symbols can use 8-bit JIS codes and shift JIS codes other than the single-byte special characters (except for the underscore and question mark).
A symbol cannot begin with a single-byte digit.

(2) A symbol can be between 1 and 253 characters in length (for single-byte characters). If a symbol exceeds 253 characters in length, an S error (syntax) error occurs. Each shift JIS code character consists of two bytes.

(3) A label must be terminated with a single-byte colon (:) (3AH). (A single-byte space or TAB code may be inserted between the label and the colon.)

(4) When using the EQU, SET, or MACRO pseudo instruction, a name must be entered in the symbol field. The name must be terminated with a single-byte space or TAB code.

(5) A symbol cannot be defined more than once. Otherwise, an S error (symbol duplication error) occurs. This does not apply to the symbols[Note] defined in the SET pseudo instruction or to those defined in macros that are not declared as being global.
If not declared public, symbols having identical names can be used in different modules; the system regards them as being different symbols.

(6) Reserved words cannot be defined as symbols. It is possible to define symbols containing reserved words.

(7) Symbols are case-sensitive.

[Example 1]

Valid	Invalid
`FIF4:`	`1F4F:`...............Begins with a digit.
`LABEL:`	`LABEL` ....Does not end with a colon (for a label).
`HERE:`	`HE RE:`............A blank is embedded in the symbol.
`ANH:`	`ANL:`.................Instructions cannot be used as symbols.
`ENDX:`	`END:`.................Pseudo instructions cannot be used as symbols.

[Example 2]

```
ABC    EQU    3
XYZ    EQU    ABC
```

The same data "3" is assigned to both ABC and XYZ.

Note The value of the symbol defined in a SET pseudo instruction can be changed. To change the value, use a SET pseudo instruction.

2.6 Mnemonic Field

Enter an instruction, pseudo instruction, or control instruction in the mnemonic field.

For an instruction requiring an operand, a blank (one or more single-byte spaces or a TAB code) is required to delimit the mnemonic field from the operand field.

[Example]

Valid	Invalid
`JMP LOOP`	`JMPLOOP`.......... A blank is not inserted between the mnemonic and operand fields.
`RET`	`RE T`................ A blank is inserted in the mnemonic field.
`SCAF`	`SCA`.................. The SCA instruction is not supported by the μPD6133.

2.7 Operand Field

In the operand field, enter the data (operand) necessary to execute the instruction. Some instructions do not require operands while others require one or two operands.

When two operands are required, delimit the operands with a comma ",".

A blank is required between the mnemonic and operand fields.

2.7.1 Operand field entry types

(1) Constant

Constants include numeric constants that consist of digits, or character constants that consist of characters.

Numeric constants include binary, octal, decimal and hexadecimal constants, all consisting of single-byte digits.

(a) Binary constant

A single-byte character "B" must be added to the end of a binary string.

(b) Octal constant

A single-byte "O" or "Q" must be added to the end of an octal string.

(c) Decimal constant

A single-byte "D" must be added to the end of a decimal string. The "D" can be omitted.

(d) Hexadecimal constant

A single-byte "H" must be added to the end of a hexadecimal string. If a constant begins with a single-byte character other than 0 to 9, a "0" must be added to the beginning of the constant.

(e) Character constant

A character constant is a string of 8-bit JIS character codes (except the LF code) and shift JIS character codes, enclosed in single-byte quotation marks (').

With AS6133, character constants can be used in the TITLE and INCLUDE pseudo instructions only.

As a result of assembly, the characters enclosed in single-byte quotation marks are converted to 8-bit JIS or shift JIS codes.

To use a single-byte quotation mark in a character constant, it must be enclosed in quotation marks.

No operations can be performed on character constants.

[Example]

'A' 41H
(Single byte)
'Ａ' 8260H
(Double byte)
'''' 27H
(Single byte) When two quotation marks are written, one single-byte quotation mark is reserved as a constant.
'A''' 4127H
(Single byte)
' ' 20H
(Single-byte space)
'＜' 203CH
'日' 93FAH
'日本電気' 93FA967B93648B43H

(2) $ (location counter)

"$" indicates the value of the location counter. In other words, it indicates the program memory address of the instruction for which the "$" is used.

[Example]

```
Address
 101            MOV     A,R11
 102    LOOP:   INC     A
 103            JNC     $-1
 105            JMP     $+20H
```

The "$" in "JNC $–1" indicates address 103H. Consequently, $–1 indicates address 102H. The $ in "JMP $+20H" indicates address 105H.

"JNC $–1" is equivalent to "JNC LOOP" where LOOP is a label.

(3) Symbol

When a symbol is entered in an operand field, the value assigned to the symbol (label or name) is assumed as the operand value.

[Example 1]

```
A1:       JC       A2
           :
A2:       ANL      A,R10
```

[Example 2]

```
VALUE   EQU    1H
        MOV    A,#VALUE
```

"MOV A,#VALUE" is equivalent to "MOV A,#1H".

(4) Expression

An expression (character or numeric expression) consists of constants, $, and symbols that are combined with operators. There are seventeen operators (+, –, *, /, MOD, NOT, AND, OR, XOR, SHR, SHL, EQ or =, NE or <>, GT or >, GE or >=, LT or <, and LE or <=). The priorities of these operators are predetermined. For details, see **Section 2.9**.

2.8 Comment Field

A comment field begins with a single-byte semicolon (;), followed by the comment itself. Comments assist the programmer in understanding the program when he or she refers to the assembly listing. While they are displayed on the assembly listing, the assembler ignores them.

A comment can be written all 8-bit JIS codes (except the LF code) and shift JIS codes.

When consecutive semicolons (;;) are used in a macro definition, the assembler handles the comment as a comment within the macro definition. It does not print the comment during macro expansion.

A comment must be terminated with an LF code. If a comment is too long to fit on one line, start the next line with a semicolon (;).

2.9 Expressions and Operators

2.9.1 Expressions

An expression (character or numeric expression) consists of constants, $, symbols, and operators in an operand field.

2.9.2 Overview of operators

(1) Overview

The operators of AS6133 assembly language are divided into five types. The priorities of these operators are predetermined.

1. Arithmetic operators
   +, –, *, /, and MOD
2. Logical operators
   OR, AND, XOR, and NOT
3. Comparison operators
   EQ, NE, LT, LE, GT, and GE
   (or =, <>, <, <=, >, and >=)
4. Shift operators
   SHR and SHL
5. Others

(and)	(operation order specifiers)
&	(replacement operator)

(2) Operator priorities

The priorities of the operators are predetermined as listed in the table. Enclosing an operator in (and) allows the order in which operations are performed to be changed.

When multiple operators having the same priority exist in a single expression, they are executed in order, from left to right.

In the table below, the highest priority is indicated by 1.

Table 2-1. Operator Priorities

Priority	Operators
1	( ) (Operation order specifiers)
2	*, /, MOD, SHL, SHR
3	+, –
4	EQ, NE, LT, LE, GT, GE, =, <>, <, <=, >, >=
5	NOT
6	AND
7	OR, XOR

2.9.3 Arithmetic operators

(1) Addition operator (+)

Adds operands together.

[Example]

Address	Symbol	Mnemonic	Operand
0010	START:	JMP	$+6

The JMP instruction causes a jump to address 16H.

(2) Subtraction operator (–)

Subtracts one operand from another.

[Example]

Address	Symbol	Mnemonic	Operand
0020	BACK:	JMP	BACK–6

The JMP instruction causes a jump to address 1AH.

(3) Multiplication operator (*)

Multiplies operands.

[Example]

Address	Symbol	Mnemonic	Operand
	A6:	MOV	A,#(2*3)

The MOV instruction causes 6 (2*3) to be loaded into Acc.

(4) Division operator (/)

Divides one operand by another. The remainder, if any, is truncated.

[Example]

Address	Symbol	Mnemonic	Operand
	A5:	MOV	A,#(256/50)

The MOV instruction causes 5 (256/50) to be loaded into Acc.

(5) MOD operator

Finds the remainder resulting from an operand division.

[Example]

Address	Symbol	Mnemonic	Operand
	A5:	MOV	A,#MOD

The MOV instruction causes 6 (remainder of division 256/50) to be loaded into Acc.

2.9.4 Logical operators

(1) OR operator

Finds the OR of operands.

[Example]

Address	Symbol	Mnemonic	Operand
	`MDFY1:`	`MOV`	`A,#(0AH OR 5H)`

The MOV instruction causes 0FH to be loaded into Acc.

(2) AND operator

Finds the AND of operands.

[Example]

Address	Symbol	Mnemonic	Operand
	`MASK:`	`MOV`	`A,#(1AH AND 0FH)`

The MOV instruction causes 0AH to be loaded into Acc.

(3) XOR operator

Finds the exclusive OR of operands.

[Example]

Address	Symbol	Mnemonic	Operand
	`MDFY2:`	`MOV`	`A,#(9AH XOR 9DH)`

The MOV instruction causes 7H to be loaded into Acc.

(4) NOT operator

Finds the 1's complement of the value of an operand.

[Example]

Address	Symbol	Mnemonic	Operand
	`COMPL:`	`MOV`	`A,#(NOT 3H AND 0FH)`

The MOV instruction causes 0CH (when NOT 3H; 0FFFCH, AND 0FH) to be loaded into Acc.

2.9.5 Comparison operators

(1) EQ (EQual) operator

Returns 0FFFFH if the values on the right and left sides are equal; otherwise, returns 0.
"EQ" can be replaced by "=".

[Example]

Address	Symbol	Mnemonic	Operand
	`COMP1:`	`MOV`	`A,#(ONE EQ 1)`

The MOV instruction causes 0FH to be loaded into Acc if ONE is 1 and 0 if ONE is other than 1.

(2) NE (Not Equal) operator

Returns 0FFFFH if the value on the left side is not equal to that on the right side; otherwise, returns 0.
"NE" can be replaced by "<>".

[Example]

Address	Symbol	Mnemonic	Operand
	`COMP2:`	`MOV`	`A,#(ONE NE 1)`

The MOV instruction causes 0FH to be loaded into Acc if ONE is not 1; loads 0 if ONE is 1.

(3) LT (Less Than) operator

Returns 0FFFFH if the value on the left side is less than that on the right side; otherwise, returns 0.
"LT" can be replaced by "<".

[Example]

Address	Symbol	Mnemonic	Operand
	`COMP3:`	`MOV`	`A,#(MINI LT 5)`

The MOV instruction causes 0FH to be loaded into Acc if MINI is less than 5, or 0 if MINI is equal to or greater than 5.

(4) LE (Less Than or Equal) operator

Returns 0FFFFH if the value on the left side is equal to or less than that on the right side; otherwise, returns 0.
"LE" can be replaced by "<=".

[Example]

Address	Symbol	Mnemonic	Operand
	`COMP4:`	`MOV`	`A,#(MINI LE 5)`

The MOV instruction causes 0FH to be loaded into Acc if MINI is equal to or less than 5, or 0 if MINI is greater than 5.

(5) GT (Greater Than) operator

Returns 0FFFFH if the value on the left side is greater than that on the right side; otherwise, returns 0.
"GT" can be replaced by ">".

[Example]

Address	Symbol	Mnemonic	Operand
	COMP5:	MOV	A,#(MAX GT 5)

The MOV instruction causes 0FH to be loaded into Acc if MAX is greater than 5, or 0 if MAX is equal to or less than 5.

(6) GE (Greater Than or Equal) operator

Returns 0FFFFH if the value on the left side is equal to or greater than that on the right side; otherwise, returns 0.
"GE" can be replaced by ">=".

[Example]

Address	Symbol	Mnemonic	Operand
	COMP6:	MOV	A,#(MAX GE 5)

The MOV instruction causes 0FH to be loaded into Acc if MAX is equal to or greater than 5, or 0 if MAX is less than 5.

2.9.6 Shift operators

(1) SHR (Shift Right) operator

Shifts the value on the left side to the right by the value on the right side.
As a result of the shift, the MSB is set to 0.

[Example]

Address	Symbol	Mnemonic	Operand
01FA	FIELD:	MOV	A,#($ SHR 5)

$ (address: 01FAH) is shifted to the right by five bits.
As a result, 0DH is loaded into Acc.

(2) SHL (Shift Left) operator

Shifts the value on the left side to the left by the value on the right side.
As a result of the shift, the LSB is set to 0.

[Example]

Address	Symbol	Mnemonic	Operand
0021	FLY:	JMP	FLY SHL 2

FLY (address: 0021H) is shifted to the left by two bits.
As a result, control jumps to address 0084H.

2.9.7 Other operators

(1) ( ) (operation order specifiers)

Indicate that the operation(s) enclosed within the parentheses should be performed first, irrespective of the operator priorities.

The parentheses ( ) can be nested to up to 16 levels.

[Example]

```
5+8–6*2/4                          = 10
5+(8–6)*2/4                        = 6
(5+8–6)*2/4                        = 3
2*(0FH–(0BH AND (0AH OR 0FH)))     = 8
2*0FH–0BH AND 0AH OR 0FH           = 0FH
```

(2) & (replacement) operator

Used in a macro definition statement to concatenate the characters on the two sides of & during macro expansion. The & itself is replaced with a NULL code.

[**Example**]

```
MOVI    MACRO X
        MOV A,#&X
        ENDM
MOVI    1
MOV    A,#1
```

CHAPTER 3 PSEUDO INSTRUCTIONS AND CONTROL INSTRUCTIONS

3.1 Outline of Pseudo Instructions and Control Instructions

The basic function of the assembler is to convert instructions into machine language. Pseudo instructions and control instructions are provided to enhance the assembler's ease of use, as well as the readability of the output listings.

Pseudo instructions and control instructions are not converted to machine language. Instead, they are used to direct the operation of the assembler. An exception to this, however, is the built-in macro pseudo instructions which are converted to machine language.

3.2 Pseudo Instructions

AS6133 mnemonic field can contain a pseudo instruction.

(1) Location counter control pseudo instruction

- **ORG**

(2) Symbol definition pseudo instruction

Symbol definition pseudo instructions are used to define an arbitrary numeric, data memory address, flag, or label.

- **EQU**
- **SET**

Values assigned by symbol definition pseudo instructions cannot be changed. However, a symbol that has already been defined by a SET pseudo instruction can be changed by using another SET pseudo instruction. Therefore, the SET pseudo instruction is used to define a variable that is significant only at assembly time.

(3) External definition and external reference pseudo instructions

External definition and external reference pseudo instructions define and reference a symbol that is used by more than one module.

- **PUBLIC-BELOW-ENDP** (external definition pseudo instruction)
- **EXTRN** (external reference pseudo instruction)

(4) Data definition pseudo instruction

Data definition pseudo instructions are used to define data in a table area.

- **DW:** Defines 8-bit data.
- **DT:** Defines 10-bit timer table data.

(5) Conditional assembly pseudo instruction

The effective use of the conditional assembly pseudo instruction enables efficient programming and, furthermore, allows a library of source programs to be created.

- **IF-ELSE-ENDIF**

(6) Repetitive pseudo instruction

The effective use of the repetitive pseudo instruction enables efficient programming.

- **REPT-(EXITR)-ENDR**

(7) Macro definition pseudo instruction

When a particular routine is used several times within a single program, a subroutine is usually used to save the number of program steps. When there are several similar processing routines having different parameters, such that a subroutine cannot be applied, a macro function is used to improve programming efficiency.
The macro definition pseudo instruction is used to define such a macro. See **Section 3.5** for details.

- **MACRO-ENDM**

(8) Global declaration pseudo instruction for symbols in a macro

- **GLOBAL**

(9) Assembly termination pseudo instruction

The assembly termination pseudo instruction indicates the end of a source (program) module.

- **END**

(10) Mask option specification pseudo instruction

- **OPTION-ENDOP**

3.3 Control Instructions

With AS6133, a mnemonic field can contain a control instruction. Control instructions are not converted to machine language. Instead, they control the output list format and source input after assembly.

Control instructions are valid only within the modules in which they are used.

(1) Output list control instructions

Output list control instructions are used to enhance the readability of the assembly listing.

- **TITLE:** Prints a title for the assembly listing.
- **EJECT:** Invokes a page change.

(2) Source input control instruction

When a program (source module) file becomes overly large, such that the programmer decides to divide the file, the source input control instruction can be used. The source input control instruction can also be used to enable the use of a previously created program (a program in a library).

- **INCLUDE**

A file can be referenced by using the INCLUDE control instruction with the relevant file name specified.

3.4 Pseudo Instructions and Control Instructions

This section explains each of the pseudo instructions and control instructions listed below.

Table 3-1. Pseudo Instructions and Control Instructions

Instruction	Name		Page
Pseudo instructions	ORG	Location counter control pseudo instruction	p.46
	EQU	Symbol definition pseudo instruction	p.47
	SET		p.48
	PUBLIC-BELOW-ENDP	External definition pseudo instruction	p.49
	EXTRN	External reference pseudo instruction	p.51
	DW	Data definition pseudo instruction	p.52
	DT		p.53
	IF-ELSE-ENDIF	Conditional assembly pseudo instruction	p.54
	REPT-(EXITR)-ENDR	Repetitive pseudo instruction	p.55, p.56
	MACRO-ENDM	Macro definition pseudo instruction	p.57
	GLOBAL	Global declaration pseudo instruction for symbols in a macro	p.59
	END	Assembly termination pseudo instruction	p.60
	OPTION-ENDOP	Mask option specification pseudo instruction	p.61
	USEPOC/NOUSEPOC		p.62
	USECAP/NOUSECAP		p.63
Control instructions	TITLE	Output list control instructions	p.64
	EJECT		p.65
	INCLUDE	Source input control instruction	p.66

ORG ORIGIN ORG

Symbol	Mnemonic	Operand	Comment
[label:]	ORG	<expression>	[;comment]

[Function]

Sets a value in the location counter.

[Usage]

(1) The ORG pseudo instruction specifies the start address of program memory. Code an ORG pseudo instruction at the beginning of each segment.

(2) The ORG pseudo instruction specifies the start address of a table area. When this instruction is specified, any change made before the table area address has no effect on the table area address.

[Explanation]

(1) Before a symbol can be used as an expression in the operand field, that symbol must have been defined.

(2) Unless an address is specified with the ORG pseudo instruction at the beginning of a program, the assembler assigns address 0000 to the location counter.

(3) If the address value specified with an ORG pseudo instruction is smaller than the previous location counter value, an A error (address specification error) occurs. If such an error occurs, the evaluation value coded in the operand is ignored, with the consecutive value next to the location counter value that existed immediately before the ORG instruction being assumed.

(4) The previous location counter value is assigned to the label added to the ORG pseudo instruction.

[Example]

```
015D            INC     A
015E            RET
0200    STRT:   ORG     200H
0200            MOV     A,#1
```

Label STRT is assigned 15FH. The operand of the ORG pseudo instruction is 200H, so the MOV instruction is assigned to address 200H.

EQU EQUATE EQU

Symbol	Mnemonic	Operand	Comment
name	EQU	<expression>	[;comment]

[Function]

Assigns the value of the expression specified in the operand to the name coded in the symbol field.

[Usage]

This instruction is used to define a data memory address.

[Explanation]

(1) Before a symbol can be coded in the operand field, the symbol must first be defined.

(2) Delimit the symbol field, mnemonic field, and operand field with a blank.

(3) If the symbol or mnemonic field contains an error, the specified name is not registered. Accordingly, a statement referencing that name becomes invalid. If the operand contains an error, 0 is assigned to the name.

(4) For a name defined using the EQU pseudo instruction, redefinition within the module in which the name is defined is not possible. If an attempt is made to redefine the name, an S error (duplicate symbol definition) occurs.

(5) When a name is defined using the EQU pseudo instruction, the name can be referenced by an instruction prior to the definition only when the name is specified in the operand of the instruction.

(6) The defined expression value is not converted to μPD6133 code; the value is assigned as is.

[Example]

```
P3_INIT     EQU     12H
P3_MOD      EQU     P3_INIT
            ;
            OUT     P3,#P3_MOD
```

SET SET SET

Symbol	Mnemonic	Operand	Comment
name	SET	<expression>	[;comment]

[Function]

Assigns the value of the expression coded in the operand field to the name in the symbol field.

In the operand field, memory name R_0 to R_F, R_{10} to R_{1F}, or R_{00} to R_{0F} can be coded in addition to an expression.

[Usage]

This instruction is used to set a formal parameter for a conditional assembly pseudo instruction (IF-ELSE-ENDIF) or repetitive operation pseudo instructions (REPT-ENDR, EXITR).

[Explanation]

(1) Delimit the symbol field, mnemonic field, and operand field with a blank.

(2) If the symbol or mnemonic field contains an error, the specified name is not registered. Accordingly, any statement referencing that name becomes invalid. If an operand contains an error, 0 is assigned to the name.

(3) For a name defined with a SET pseudo instruction, a different value can be redefined. The value defined with a SET pseudo instruction remains valid until the next SET pseudo instruction is encountered.

(4) When a name is defined using the SET pseudo instruction, the name can be referenced by an instruction prior to the definition only when the name is specified in the operand of the instruction.

(5) The defined expression value is not converted to μPD6133 code; the value is assigned as is.

[Example]

```
IMMED    SET     5
         ANL     A,#IMMED    ;IMMED=5
         ;
IMMED    SET     6
         ANL     A,#IMMED    ;IMMED=6
```

PUBLIC	**PUBLIC**	**PUBLIC**
BELOW	**BELOW**	**BELOW**
ENDP	**END PUBLIC**	**ENDP**

Format 1

Symbol	Mnemonic	Operand	Comment
[label:]	PUBLIC	<symbol-group>	[;comment]

Format 2

Symbol	Mnemonic	Operand	Comment
[label:]	PUBLIC	BELOW	[;comment]
[name	EQU	<expression (EQU-type)>]	[;comment]
	ENDP		

[Function]

The external definition pseudo instruction can be coded in either of two formats.

Format 1 is used to declare that symbols coded in the operand field are referenced by other modules.

Format 2 is used to declare that symbols defined in the block enclosed between PUBLIC BELOW and ENDP are referenced by other modules.

[Usage]

The external definition pseudo instruction declares symbols as being referenced by other modules.

[Explanation]

(1) An external definition pseudo instruction can be coded anywhere within a source program.

(2) When format 1 is used, the symbols specified for public declaration in a module must be defined using a symbol definition pseudo instruction within the same module. If a symbol coded in the external definition pseudo instruction of format 1 is not defined in the same module, an S error (Undefined Symbol) occurs.

(3) For format 2, if the block enclosed between PUBLIC BELOW and ENDP contains an instruction other than the symbol definition pseudo instructions, an S error (Syntax Error) occurs.

(4) Each statement is terminated by an LF code. If there are too many symbols to fit on one line, declare PUBLIC again on the next line.

(5) If the ENDP corresponding to PUBLIC BELOW is missing, a P error (No ENDP Statement) occurs at the END pseudo instruction.

(6) If a symbol declared as PUBLIC is not referenced by any external module, a warning (Unreference Symbol) occurs at link time.

PUBLIC	**PUBLIC**	**PUBLIC**
BELOW	**BELOW**	**BELOW**
ENDP	**END PUBLIC**	**ENDP**

[Example]

```
          PUBLIC     VAL1,VAL2
            .
            .
            .
VAL1      EQU        1
VAL2      EQU        2
            .
            .
            .
          PUBLIC     BELOW
VAL3      EQU        3
VAL4      EQU        4
          ENDP
```

EXTRN EXTERN EXTRN

Symbol	Mnemonic	Operand	Comment
[label:]	EXTRN	<symbol-group>	[;comment]

[Function]

Declares that the symbols coded in the operand field (for which public declaration is performed in other modules) are referenced in the module.

[Usage]

When symbols declared as public symbols in other modules are needed in a module, the EXTRN pseudo instruction can be used to enable the use of these symbols in the module.

[Explanation]

(1) In a module, symbols declared with the EXTRN pseudo instruction cannot be referenced before EXTRN has been specified.

(2) If a symbol for which EXTRN declaration is performed in a module is defined in the same module, an S error (Symbol Multi Defined) occurs.

DW DEFINE WORD DW

Symbol	Mnemonic	Operand	Comment
[label:]	DW	<expression>	[;comment]

[Function]

Sets an expression or characters coded in the operand field in the location indicated by the current location counter value (program memory address) as 8-bit object code.

[Usage]

This instruction is used to define 8-bit data in a table area.

[Explanation]

(1) A single expression that can be represented by eight bits can be coded for <expression>. If the value of the expression exceeds 10 bits, a V error (invalid value) occurs. If either bit 8 or 9 is 1, a warning message is generated. (In this case, bits 8 and 9 of the object code are set to 0.) If more than one expression is coded in the operand field, an O error (the number of operands is invalid) occurs.

(2) If an undefined symbol is coded in the operand field, an S error (Undefined Symbol) occurs.

(3) If the expression coded in the operand field is invalid, NOP (E0E0H) is generated as the object code.

Caution The DW instruction is used to reference table areas other than the timer table area (MOV T,@R0). To perform timer table area reference, use the DT instruction.

[Example]

```
E.      LOC.      OBJ.      M I      SOURCE STATEMENT
                  E2E0               DW      20H
                  E4E0               DW      340H              ;<1>
```

In **<1>**, a warning is generated, and a value for which bits 8 and 9 are 0 is set as the object code.

DT DEFINE TIMER DT

Symbol	Mnemonic	Operand	Comment
[label:]	DT	<expression>	[;comment]

[Function]

Sets the expression or characters coded in the operand field in the location indicated by the current location counter value (program memory address), as a 10-bit object code.

[Usage]

This instruction is used to define timer data in a table area.

[Explanation]

(1) A single expression that can be represented using 10 bits can be coded for <expression>. If the value of the expression exceeds 11 bits, a V error (invalid value) occurs. If more than one expression is coded in the operand field, an O error (invalid number of operands) occurs.

(2) If an undefined symbol is coded in the operand field, an S error (Undefined Symbol) occurs.

(3) If the expression coded in the operand field is invalid, NOP (E0E0H) is generated as the object code.

Caution The DT instruction causes object code conversion to reference a timer table area (MOV T,@R0). Therefore, never use the DT instruction for an ordinary table reference instruction. (For ordinary table reference instructions, use the DW instruction.)

[Example]

```
E.      LOC.      OBJ.       M I      SOURCE STATEMENT
                                      ;** TIME DATA **
                  F8F7                DT      21FH              ;CARRY ON
                  F1F7                DT      05FH              ;CARRY OFF
```

IF ELSE ENDIF

Symbol	Mnemonic	Operand	Comment
[label:]	IF	<expression>	[;comment]
[statement			]
	[ELSE]		[;comment]
[statement			]
	ENDIF		[;comment]

[Function]

If the value of the operand field of the IF statement is other than 0 (false), the statements enclosed between IF and ELSE are to be assembled. The statements between ELSE and ENDIF are not assembled.

If the evaluation value of the operand field of the IF statement is 0 (false), the statements enclosed between IF and ELSE are not assembled. The statements between ELSE and ENDIF are assembled, however.

[Usage]

This instruction is used in an arbitrary routine in a program to select the statements to be expanded according to the use condition of the routine.

[Explanation]

(1) All the statements between an IF and the corresponding ENDIF are defined as an IF-ENDIF block.

(2) ELSE is optional. It need not be specified. When ELSE is specified, however, it can be used only once for an IF-ENDIF block. If ELSE is specified more than once for a single IF-ENDIF block, an S error (syntax error) occurs for the second and subsequent ELSEs.

(3) Before a symbol can be coded in the operand field of the IF statement, the symbol must first be defined.

(4) Up to 40 levels of nesting, including macro reference statements and REPT statements, are possible.

(5) The ELSE and ENDIF statements cannot have any label.

[Example]

```
IF      ZZZ0      EQ  0
        NOP
        HALT      #3H
ELSE
        NOP
        HALT      #ZZZ0
ENDIF
```

REPT ENDR — REPEAT END REPEAT — REPT ENDR

Symbol	Mnemonic	Operand	Comment
[label:]	REPT	<expression (EQU-type)>	[;comment]
[statement			]
	[EXITR]		[;comment]
[statement			]
	ENDR		[;comment]

[Function]

Expands the statement enclosed between REPT and ENDR as many times as the value of <expression (EQU-type)>.

If EXITR is encountered between REPT and ENDR, expansion is terminated, and assembly is performed from the statement next to ENDR.

[Usage]

This instruction is used to repeat the same statement.

To disable the repetitive pseudo instruction temporarily or interrupt it during debugging, insert EXITR.

[Explanation]

(1) Up to eight levels of nesting are possible. When macro reference statements and IF statements are included, up to 40 levels are possible.

(2) Before a symbol can be coded in <expression (EQU-type)>, the symbol must have already been defined. If the coded symbol is not defined or was defined on a previous page, an S error (Undefined Symbol) occurs.

(3) A symbol in the operand of a pseudo instruction specified in the REPT-ENDR block must have already been defined. If the symbol is defined after the appearance of the symbol, or if the symbol is not defined, an S error (Undefined Symbol) occurs.

(4) If the ENDR corresponding to REPT is missing, a P error (No ENDR Statement) occurs for the END pseudo instruction which appears at the end of the module.

[Example]

```
REPT      3       ;Repeat the DW 0 instruction three times.
DW        0
ENDP
```

EXITR EXIT REPEAT EXITR

Symbol	Mnemonic	Operand	Comment
	EXITR		[;comment]

[Function]

EXITR in the REPT statement ends expansion, and performs assembly from the statement subsequent to ENDR.

[Explanation]

(1) The EXITR pseudo instruction can be used only between REPT and ENDR.
(2) If EXITR is coded outside the REPT-ENDR block, a P error (invalid EXITR statement) occurs.

MACRO ENDM — MACRO END MACRO — MACRO ENDM

Symbol	Mnemonic	Operand	Comment
name	MACRO	<formal-parameter-group>	[;comment]
[statements (macro-body)			]
	ENDM		

[Function]

Assigns a macro name, indicated by name, to the sequence of statements (macro body) enclosed between MACRO and ENDM.

The name is used as the definition name at macro reference time.

[Usage]

This instruction is used for macro definition.

[Explanation]

(1) Macro body

The macro body consists of symbols, instructions, pseudo instructions (except MACRO and ENDM), comments, and other macro statements including their macro bodies.

(2) Formal parameter group

- Up to 32 formal parameters, delimited by a comma (,), can be coded, using up to 253 characters.
- Formal parameters can be used only within a macro body.
- Actual parameters are assigned to the formal parameters, coded in the macro body, when the macro is referenced.
- Formal parameters can be coded in the symbol field, mnemonic field, and operand field.

(3) When two semicolons (;;) appear successively in the macro body, the subsequent character string is treated as a comment in the macro. It is not expanded when the macro is referenced.

MACRO	**MACRO**	**MACRO**
ENDM	**END MACRO**	**ENDM**

[Example 1] Macro having no parameter

```
ADDR01      MACRO                   ;Macro definition
            MOV         A,R01
            INC         A
            MOV         R01,A
            ENDM
```

[Example 2] Macro having a parameter

```
ADDRNO      MACRO       RNO         ;Macro definition
            MOV         A,RNO
            INC         A
            MOV         RNO,A       ;;RNO+1
            ENDM
            ADDRNO      R10         ;Macro reference
                 ↓  (Expansion)
            MOV         A,R10
            INC         A
            MOV         R10,A
```

[Description]

As shown in the above example, when a parameter is coded in the operand field of a macro, the parameter can be replaced by the parameter specified at the time of macro reference. A parameter in a macro definition statement is called a formal parameter.

R10 is assigned to formal parameter RNO.

Two successive semicolons (;;) are followed by a comment in the macro. This comment is not expanded at the time of reference.

GLOBAL

Symbol	Mnemonic	Operand	Comment
[label:]	GROBAL	<symbol-group>	[;comment]

[Function]

Declares symbols used in a macro as symbols that can be referenced outside the macro.

[Usage]

Before symbols used in a macro can be used outside that macro, the GLOBAL pseudo instruction must first be specified.

[Explanation]

(1) The GLOBAL pseudo instruction can be used only inside a macro definition (within the block enclosed between MACRO and ENDM). If a GLOBAL pseudo instruction is used outside a macro definition, an M error (Invalid Mnemonic) occurs.

(2) The global declaration for a symbol must be coded before that symbol is defined. If the GLOBAL declaration is performed after the symbol is defined, an S error (Symbol Multi Defined) occurs.

(3) When a symbol is declared as a global symbol in a source module program, the symbol can be used in the same source module program.

(4) One or more symbol names can be specified in the operand field of the GLOBAL pseudo instruction, provided they fit on one line (255 characters maximum).
If the length of a statement exceeds 255 characters, an S error (Syntax Error) occurs, and the statement is ignored.

[Example]

```
OBJ.   M I   SOURCE STATEMENT
             STMAC   MACRO             ;Macro definition
                     GLOBAL   SYMA     ;Global declaration
             SYMA    SET      00H
0000                 DW       SYMA
                     ENDM
             ;
             STMAC                     ;Macro reference
             ;
0000         DW      SYMA              ;A local symbol is referenced outside the macro.
```

[Description]

When global declaration is performed for a symbol in a macro, the symbol value can be used as is upon the completion of macro expansion.

END END END

Symbol	Mnemonic	Operand	Comment
[label:]	END		

[Function]

Directs the assembler to terminate the source (program) module.

[Usage]

Code this instruction on the last line of a source (program) module.

[Explanation]

(1) If the END pseudo instruction is not followed by the LF code (8-bit JIS code: 0AH), an error occurs.
When the screen editor is used for programming, modules can be cataloged even if the LF code is missing. Therefore, be particularly careful not to forget the LF code.

(2) If END is followed by a code other than the CR/LF code, such as a comment, a warning message is generated. Such a statement is ignored.

(3) If a source file does not end with an END statement, or if the END statement is not followed by a valid code, such as a CR code, preventing the assembler from recognizing the END pseudo instruction, a P error (END statement missing) occurs. If such an error occurs, the assembler generates an object file, assuming the END statement to be placed at the end of the file.

[Example]

```
     .
     .
     .
     .
    END
```

[Description]

In the above example, the END pseudo instruction is placed in the last line of a source program module.

OPTION ENDOP

Symbol	Mnemonic	Operand	Comment
[label:]	OPTION ENDOP	mask-option-pseudo-instruction	[;comment]

[Function]

The block enclosed between OPTION and ENDOP is called a mask option definition block. In the mask option definition block, a mask option pseudo instruction can be coded. The mask option pseudo instruction varies depending on the device.

[Explanation]

(1) The OPTION pseudo instruction must be terminated by the ENDOP pseudo instruction. If an END pseudo instruction appears between the OPTION and ENDOP pseudo instructions, a P error (No OPTION Directive) occurs.

(2) If an instruction that generates an object code is placed between the OPTION and ENDOP pseudo instructions, a warning is generated. In this case, the object code for the instruction between the OPTION and ENDOP pseudo instructions is not generated.

(3) An OPTION and ENDOP pseudo instruction pair can be coded only once within a source program. If they are coded more than once, a P error (Duplicated OPTION Directive) occurs for the second OPTION pseudo instruction. At this time, the object code between OPTION and ENDOP is not generated.
The OPTION and ENDOP pseudo instructions cannot be coded separately in two different modules.

(4) If a source program for a device which requires a mask option contains no OPTION pseudo instruction, an O error (Not Found Mask Option Block) occurs at link time.

[Example]

```
OPTION              ;Include a low-voltage detection circuit.
USEPOC
ENDOP
```

★

USEPOC NOUSEPOC

Symbol	Mnemonic	Operand	Comment
		USEPOC NOUSEPOC	[;comment]

[Function]

Specifies whether a low-voltage detection circuit is used by mask option.

USEPOC specifies that the low-voltage detection circuit is used, and NOUSEPOC specifies that the circuit is not used.

[Caution]

An error occurs unless either USEPOC or NOUSEPOC is specified.

★

USECAP NOUSECAP

Symbol	Mnemonic	Operand	Comment
		USECAP NOUSECAP	[;comment]

[Function]

Specifies whether a capacitor for oscillator is used by mask option.

USECAP specifies that the capacitor is used, and NOUSECAP specifies that the capacitor is not used.

[Caution]

An error occurs unless either USECAP or NOUSECAP is specified.

This option can be specified with the products D67, D68, and D69.

TITLE TITLE TITLE

Symbol	Mnemonic	Operand	Comment
[label:]	TITLE	'character-string'	[;comment]

[Function]

Causes a page feed in the assembly listing, and outputs the character string specified in the operand field in the header line of the assembly listing.

[Usage]

This instruction is used to print a title for the assembly listing and to enhance readability.

[Explanation]

(1) Up to 78 characters (8-bit JIS code) can be coded as the character string. If the character string is longer than 78 characters, an I error (invalid data length) occurs.

(2) When the TITLE control instruction appears, the assembler performs a page feed, then prints the specified title (characters) as the header. When the TITLE control instruction appears in the first line, however, the assembler does not perform a page feed. When a page feed is performed by the TITLE control instruction, the TITLE control instruction is output on the first line of a new page.

(3) If a character string is not enclosed in quotation marks ('), an S error (syntax error) occurs.

[Example]

Source program listing

```
          .
          .
          .
      TITLE         'SUBROUTINE'
          .
          .
          .
```

EJECT EJECT EJECT

Symbol	Mnemonic	Operand	Comment
[label:]	EJECT		[;comment]

[Function]

Causes a page feed in the assembly listing.

[Usage]

This instruction is used to change the page at the beginning of a new routine. When a page feed is performed, the readability of the assembly listing is enhanced.

[Explanation]

(1) When an EJECT control instruction appears, the assembler performs a page feed.

(2) When a page feed is performed by the EJECT control statement, the EJECT control statement is printed on the page before the page feed.

[Example]

Source program listing

```
                   .
                   .
                  JMP      ABC
                                            EJECT
     DEF:
                   .
                   .
```

INCLUDE INCLUDE INCLUDE

Symbol	Mnemonic	Operand	Comment
[label:]	INCLUDE	'file-name'	[;comment]

(For details of the file naming conventions, see "Preface.")

[Function]

Reads a source program specified by file name, and processes it as part of the source program.

[Usage]

This instruction is used to include another split file.

[Explanation]

(1) A source module specified by INCLUDE can contain another INCLUDE statement. Up to eight levels of INCLUDE nesting is possible. If the nesting is performed to more than eight levels, a nest overflow error occurs.

(2) The file specified by the INCLUDE control statement must end with the EOF statement. If EOF is not specified, a warning is generated.

(3) If no extension is specified for the file name, the extension is assumed to be ASM.

(4) The file connected by the INCLUDE control instruction is not a split module. Therefore, the symbols in the original source program can be referenced as is.

(5) If a file name is not enclosed in quotation marks ('), an S error (syntax error) occurs, and this specification is ignored.

(6) A path name can be used as the file name. (Up to 64 characters can be coded as a file name.)

(7) If the file specified by file name does not exist, an F error (the include file cannot be opened) occurs.

INCLUDE **INCLUDE** **INCLUDE**

[Example 1]

Source program

```
        .
        .
        .
INCLUDE      'SUB1.ASM'
        .
        .
        .
INCLUDE      'SUB2.ASM'
        .
        .
        .
     END
```

SUB1.ASM

```
      .
      .
      .
      .
```

SUB2.ASM

```
      .
      .
      .
      .
```

INCLUDE INCLUDE INCLUDE

[Example 2]

Source module A

```
 INCLUDE       'MACROFILE.ASM'
      .
      .
      .
    END
```

Source module B

```
 INCLUDE       'MACROFILE.ASM'
      .
      .
      .
    END
```

Up to 16 INCLUDE files can be coded in one source module.
The total length of the INCLUDE file names coded in one source module must not exceed 255 characters.

Caution When the /HOST option is specified, a source file name can contain neither the drive name nor directory name.

MACROFILE.ASM

```
MAC1      MACRO    A1,A2
            .
            .
            .
          ENDM
MAC2      MACRO    B1,B2
            .
            .
            .
          ENDM
            .
            .
            .
```

[Description]

Only macros that are used in multiple modules are placed in one file. Then, the file is included by using the INCLUDE control statement. The macros can be shared by source modules without having to use the PUBLIC and EXTRN pseudo instructions. When the PUBLIC and EXTRN pseudo instructions are used, however, the names of the macros used must be declared in each module.

3.5 Macro Function

When a particular routine is used several times within a single program, a subroutine is generally used to save the number of program steps. When similar processing routines but with different parameters exist, and a subroutine cannot be applied, a macro function is used to enable efficient programming.

3.5.1 Macro definition and applicable range

(1) Macro definition

To define a macro, use the macro definition pseudo instruction (MACRO, ENDM).
When a macro is defined, formal parameters can be used.
See **Table 3-1** for details of the macro definition pseudo instruction.

(2) Applicable range of macros

Two types of symbols are defined in a macro: local symbols that can be used only within the macro, and global symbols that can be used both in the macro and in other routines.
To use symbols as global symbols, perform global declaration in the macro by using the GLOBAL pseudo instruction. Symbols that are not declared as global symbols are handled as local symbols and can be used only within the macro. See **Table 3-1** for details of the GLOBAL pseudo instruction.
When a macro is used, program readability can be improved by assigning an easily remembered name to a sequence of blocks that represents the procedure performed by those blocks. In addition, the macro can be used in much the same way as a library. To do this, create a separate file containing macro definition statements only. Then, specify the INCLUDE statement at the beginning of a source program to read the contents of the file.

- **Local symbols**

  Symbols defined in a macro are assumed to be local symbols unless declared otherwise. Local symbols can be used only within the macro in which they are defined. In this case, macro reference statements in the macro and INCLUDE statement in the macro are also included. Therefore, even when the same symbol name as that of a local symbol in a macro is defined outside the macro, or when a particular macro is referenced more than once, such that similar statements are generated, the assembler does not regard them as a duplicate definition.

- **Global symbols**

  A symbol defined in a macro sometimes needs to be referenced from outside of that macro. To do this, the symbol must be declared as a global symbol to enable the symbol to be referenced from any statement in the module in which the symbol is used. (See **Table 3-1** for an explanation of the global declaration method and an example of its use.)
  Note, however, that when a symbol defined by other than the SET pseudo instruction references a macro declared as being global in a fixed manner more than once, such that a sequence of statements is generated, a duplicate definition error occurs for that symbol.
  If a value is defined for a symbol with the SET pseudo instruction outside a macro, and the same symbol is set inside the macro, the symbol is treated as a local symbol in the macro. Then, that symbol has no relationship with the symbol having the same name but which is outside the macro. When the symbol outside the macro needs to be assigned a value in the macro, global declaration is required.

(3) Using a macro

The use of a macro requires that definition and reference be performed. Assigning a macro name to a sequence of instructions and pseudo instructions is referred to as macro definition.

[Example 1]

```
ADDR01      MACRO                       ;Macro definition
            MOV         A,R01
            INC         A
            MOV         R01,A
            ENDM
```

In the above example, macro name ADDR01 is assigned to the following three instructions:

```
MOV     A,R01
INC     A
MO      R01,A
```

An arbitrary macro name can be specified. However, the specified macro name must be neither an existing symbol name nor a reserved word.

When a macro is defined in a module, it can be used from that module any number of times after being defined.

Specifying a macro name to use the contents of the macro definition is referred to as macro reference.

Code a macro reference statement in the mnemonic field.

When a macro is referenced, the assembler expands the instructions and pseudo instructions assigned to the macro in the order in which they are defined. This is referred to as macro expansion.

[Example 2]

```
ADDR01                    ;Macro reference
  ↓ (Expansion)
MOV   A,R01
INC   A
MOV   R01,A
```

The following lists the macro-related pseudo instructions:

MACRO-ENDM
GLOBAL
REPT-EXITR-ENDR

3.5.2 Macro reference

[Function]

A macro body defined with the MACRO and ENDM statements is referenced.

[Format]

Symbol	Mnemonic	Operand	Comment
[label:]	name	<actual-parameter-group>	[;comment]

[Explanation]

(1) As the name, specify the macro name coded in the symbol field of the MACRO statement. The name must be defined before it can be referenced.

(2) The following five forms can be coded as actual parameters. They are evaluated as 16-bit data.

(a) Expression

(b) Character constant (8-bit JIS code or shifted 8 JIS code string, enclosed in quotation marks)

(c) Space or blank (no coding, comma only)

(d) Symbol

(e) Constant

(3) Formal parameters are replaced by actual parameters on a one-to-one basis in the order in which they are coded, starting from the left. If the number of actual parameters exceeds the number of formal parameters, an O error (Operand count error) occurs.

If the number of actual parameters is smaller than the number of formal parameters, a NULL code is assigned to the remaining formal parameters for which no corresponding actual parameters exist. In this case, no error occurs at macro reference time. However, an error caused by the NULL code may occur at macro expansion.

(4) When a blank, comma, quotation mark, semicolon, or tab is coded as an actual parameter, it must be enclosed in quotation marks so that it can be handled as a character string.

(5) A macro body can contain macro reference statements. Up to 40 levels of nesting, including repetitive pseudo instructions, macro reference statements, and IF statements, is possible. If the nesting depth exceeds 40 levels, an N error (Nesting overflow) occurs, and the illegal nesting is not assembled. Alternatively, an M error (Macro area overflow) occurs, and the macro is not expanded.

[Example]

```
ADMAC       2,5
```

ADMAC is a macro name defined with a macro definition pseudo instruction, and 2 and 5 are actual parameters required when ADMAC is referenced.

3.5.3 Macro expansion

The assembly of source programs using macros consists of the following steps:

<1> When a macro definition is encountered, the macro body is stored in an internal memory area of the assembler as is (macro registration).

<2> When a macro reference is found, the symbol table is searched for the corresponding macro body, after which the macro body is inserted in the macro name position.

<3> The expanded program is assembled. When two successive semicolons (;;) appear in a macro body, the portion between ;; and the end of that line is regarded as a comment, such that that portion is not expanded at macro reference.

[Explanation]

(1) Macro expansion is performed in path 1 of the module assembly phase.

(2) Before a symbol defined outside a macro can be referenced by the operand of a pseudo instruction coded in the macro, the symbol must be defined prior to the macro reference. If the symbol is not defined or is defined after the macro reference, an S error (Symbol undefined) occurs.

[Example]

```
HTIMER      MACRO     TIMEVAL,HALTVAL      }
            MOV       T,#TIMEVAL           }
            HALT      #HALTVAL             } <1>
            ENDM                           }

            HTIMER    100H,0101B           ---- <2>
```

<1>: A macro named HTIMER is defined.
TIMEVAL and HALTVAL are formal parameters.

<2>: A macro named HTIMER is referenced.
100H and 0101B are actual parameters. They correspond to formal parameters TIMEVAL and HALTVAL, respectively.

As a result of the reference to HTIMER, expansion is performed as follows:

```
MOV        T,#100H
HALT       #0101B
```

PART II

OPERATION

CHAPTER 1 PRODUCT OVERVIEW

★ 1.1 Product Description

Program Name	File Name	File Type
Assembler	AS6133.EXE	Command file

The command file is the first file to be read into memory when program execution begins.

★ 1.2 Supported Debugger

Use the following debugger when using the AS6133 assembler. NEC's SM6133 V1.02 and V1.06 cannot be used.

Manufacturer:	Naito Densei Machida Mfg.
Product name:	EB-6133 emulator

★ 1.3 System Configuration

This section describes the operating environment necessary to run AS6133.

(1) Host machine
See "Preface" for the personal computers on which this assembler can run.

(2) Operating system
★ See "Preface" for the operating systems on which this assembler can run.

(3) User memory size
512 KB or larger

(4) Files necessary to run AS6133

1. Source file (××××.ASM)
   This is a file of a source program to be assembled.

2. Sequence file (××××.SEQ)
   This is a file of information necessary to specify a device file name, assembly options, and a source file name at the start of the assembler.
   When more than one source module file is to be assembled, it is necessary to specify the source file names in the sequence file beforehand.

3. MS-DOS environment file (CONFIG.SYS)
   ★ Setting: files = 15 (15 or more)
   buffers = 10 (10 or more)

CHAPTER 2 BEFORE EXECUTION

2.1 Creating a Backup File

Before using AS6133, create its backup copy by copying the contents of the original assembler disk to a work disk. This is to prepare for disruption of the contents of the floppy disk or the disk itself.

Keep the original disk in a safe place.

Procedure to create a backup file

1. Start MS-DOS.
2. Insert the MS-DOS system disk into drive A, and a new floppy disk into drive B.
3. Format the new floppy disk in drive B using the FORMAT command and copy the system to it.

```
A>FORMAT B:/S ↵
A>
```

4. Remove the MS-DOS system disk from drive A, and insert the AS6133 original disk into drive A. Enter the COPY command to transfer AS6133.EXE from the disk in drive A to the disk in drive B.

```
A>COPY A:*.* B:/V ↵
A>
```

5. All the contents of the disk in drive A have been transferred to the disk in drive B.

```
A>DIR B:/W ↵
    AS6133.EXE
A>
```

★ ## 2.2 Install

(1) Copy the file (AS6133.EXE) in the supplied medium to the install destination.
For example, if the supplied medium is set in floppy disk drive A: and the install destination is C:\nectools\bin, execute the copy command as follows:

```
X> copy A:*.*  C:\nectools\bin
```

(2) Add the directory at the install destination to environmental variable PATH.
In the above example, add the following line to AUTOEXEC.BAT.

```
PATH  C:\nectools\bin;%PATH%
```

CHAPTER 3 SEQUENCE FILE

3.1 Overview

When starting the assembler and assembling a program, it is necessary to specify the target device file, source module file, and assembly options.Note (This information is generically called an assembly condition.)

The assembly condition is specified in a sequence file.

Using a sequence file makes it possible to specify many assembly conditions under one sequence file name.

During debugging, source module files can be deleted or added simply by changing the contents of the sequence file.

Effective use of a sequence file can make debugging efficient, as described above.

Note An assembly option specifies, for example, whether to output an assembly listing. See **Section 4.5** for details.

3.2 Sequence File Format

The sequence file is created using an editor or the COPY command.

The file extension of the sequence file must always be ".SEQ".

3.2.1 Overall format

```
[;comment]
          device-name                                          [;comment]      ; <1>
          /option[/option/option/.../.../......]               [;comment]      ; <2>
          source-file-name                                     [;comment]     ┐
                      .                                              .        │
                      .                                              .        │ <3>
                      .                                              .        │
          source-file-name                                     [;comment]     ┘
```

[Description]

(1) Specify a device name at **<1>**.

(2) Specify assembly options at **<2>**. Only one assembly option can be placed between two adjacent slashes (/). To specify more than one assembly option, write them in succession and separate them with a slash. If two or more lines are used to specify assembly options, each line must begin with a slash.
Assembly options specified at **<2>** are effective when any source file is assembled.

(3) Specify a source module file at **<3>**.

(4) In the sequence file, begin a comment with a semicolon (;) in the same way as in the source program. A comment can be placed anywhere in the sequence file.

(5) The device name, assembly options, and source file name must be specified in the stated order. Otherwise, an error is detected.

3.2.2 Device name format

```
[;comment]
  device-name                    [;comment]
```

[Function]

The device name of the product that is the target of assembly is specified.

[Description]

(1) Usually, the sequence file should begin with a device name. However, a comment can precede the device name.

(2) No file extension is used. If an unspecified device name is used, the following error is detected during assembly, and assembly is aborted.

NOT FOUND DEVICE STATEMENT

If a device name is written in a place where it should not be, an error will be detected during assembly.

★ (3) Table 3-1 shows the correspondence between device names that can be described in the sequence file and devices.

Table 3-1. Device Name That Can Be Described and Supported Device

Device Name	Supported Device
D6133	μPD6133
D6134	μPD6134
D6135	μPD6135
D6604	μPD6604
D6605	μPD6605
D63	μPD63
D63A	μPD63A
D64	μPD64
D64A	μPD64A
D62	μPD62
D62A	μPD62A
D65	μPD65
D6132	μPD6132
D6132A	μPD6132
D67	μPD67
D68	μPD68
D69	μPD69

Caution Use the device name of the corresponding ROM version device when using the μPD61P34B, 66P04B, 6P4B, 6P5, or 6P9.

[Example]

When the μPD6133 is the target product

```
D6133            ; μPD6133
```

Note Leave out "μP" from the product name.

3.2.3 Assembly option format

```
[/option]  [/option]   [/......]   [/option]
[/option]  [/......]   [/option]   [;comment]
```

[Function]

Assembly options are specified.

[Description]

(1) Usually, the specification of assembly options should begin on a line immediately after the device name file. However, a comment can precede the specification of assembly options.
(2) Each assembly option must be prefixed with a slash (/).
(3) To specify more than one assembly option, separate them with a slash. One or more space characters are allowed between two assembly options.
(4) Assembly options may be written over more than one line. Each line must end with a pair of CR/LF characters, and each continuation line must begin with a slash.
(5) If mutually exclusive assembly options are specified, the last one to appear is effective.
(6) Assembly options can be omitted.
(7) See **Section 4.5** for details of assembly options.

If an assembly option is specified in a place where it should not be, an error is detected during assembly.

3.2.4 Source file name format

```
source-file-name  [;comment]
source-file-name  [;comment]
         :
source-file-name  [;comment]
```

[Function]

The name of a source file to be assembled is specified.

[Description]

More than one source file name cannot be specified on one line.

Caution If the /HOST option is specified, neither a drive name nor a directory name can be specified in a source file name.

3.3 Example of a Sequence File Description

An example of describing a sequence file (SAMPLE.SEQ) is given below.

```
        ;DEVICE NAME
            D6134        ;  <1>
        ;OPTION          ─┐
            /HOST         │ <2>
            /WORK=B:     ─┘
        ;SOURCE MODULE   ─┐
            INIT.ASM      │
            MAIN.ASM      │
            SUB1.ASM      │ <3>
            SUB2.ASM      │
            DATA.TBL     ─┘
```

[Description]

<1> is the name of a device that is the target of assembly.

<2> is the specification of assembly options.

<3> is a source module to be assembled.

A sequence file can be created using either an editor running on MS-DOS or the COPY command (MS-DOS system command).

The COPY command may be sufficient if the sequence file to be created is small. However, if it is necessary to correct a sequence file or create a large sequence file, an editor will be more convenient.

Caution If the /HOST option is specified, neither a drive name nor a directory name can be specified in a source file name.

CHAPTER 4 ASSEMBLER FUNCTIONS

4.1 Overview

AS6133 reads a specified source module file and creates files such as an object file and assembly list file from the statements in the source module file.

AS6133 uses a two-pass assembly method. In the first pass, a symbol table is created, and mnemonics are converted to machine words. Symbols are left undefined, but an area is reserved for them.

In the second pass, the symbol area reserved in the first pass is allocated to the machine words. After the second pass ends, an intermediate object module file is created. When the intermediate object module file is created, address information about branches extending over more than one module file has not been resolved.

Next, AS6133 links the intermediate object module files to create an object file. This linkage processing is started automatically.

AS6133 has an assembly time reduction function to make assembly efficient. When an intermediate object module file is created at the end of the second pass, the date/time of creation is added to the intermediate object module file. When a source module file is partly corrected and reassembled, the creation date/time of the source module file is compared with that of the existing intermediate object module file that has the same file name as that source module file. The source module file is assembled only when its creation date/time is more recent than that of the intermediate object module file.

If the creation date/time of an object module file is more recent than that of the corresponding source module file, AS6133 assumes that the source module file has not been changed and need not be reassembled. In this case, the existing object module file is used in linkage editing.

4.2 Assembly Input/Output Files

AS6133 uses the following input files.

Input File Name	Description	File Type
Source file	Source file created using an editor	.ASM
Sequence file	File in which a device name, the specification of assembly options, and a source module file are saved. * Use of a sequence file eliminates the necessity to specify a device name, assembly options, or a source module file each time the assembler is started, thus making assembly efficient.	.SEQ

Remark An underlined file extension can be changed.

AS6133 uses the following output files.

Output File Name	Description	File Type
PROM file	File holding object code in Intel hex format, and IFL/DFL. IFL/DFL is followed by an end code in Intel hex format. The object code and IFL/DFL are written at one time by downloading the PRO file to the PROM writer.	.PRO
Assembly list file	File holding the assembly list of a source module file.	.PRN
Cross-reference list file	File holding the cross-reference list of a source module file. If no list is output, the file extension is .XRF.	.PRN
Log file	File holding error and warning messages to be output to the console during assembly. The name of this file is fixed at "AS6133.LOG."	.LOG
Intermediate object module file	Intermediate file created for each source file during assembly During linkage, the intermediate object module file is used as an input file.	.OBJ

Remark An underlined file extension can be changed.

4.3 Assembler Functions

4.3.1 Intermediate object module file output function

A source module file (.ASM) specified at the start of assembly is converted to machine words, which are then output to an intermediate object module file (.OBJ) having the same name as the source module file.

The intermediate object module file is added with the date/time it was created.

4.3.2 Linkage function

AS6133 is an absolute assembler, but it has a linkage function so that a source file split into modules can be assembled.

When source module files are assembled, an intermediate object module file is created for each source module file, and linkage is automatically carried out later by accepting the intermediate object module files as input.

4.3.3 PRO file output function

A PRO file is created by linking intermediate object module files. The PRO file consists of the object part and IFL/DFL part. It is a PROM data file for ordering a masked ROM chip.

See **Chapter 5** for details.

4.3.4 Assembly time reduction function

AS6133 has an assembly time reduction function to make debugging efficient.

Before a source module file is assembled, its creation date/time is compared with the creation date/time of an intermediate object module file having the same name as the source module file (if there is one). If the creation date/time of the intermediate object module file is more recent than that of the source module file, AS6133 assumes that the source module file has not been changed and need not be reassembled.

If the creation date/time of the intermediate object module file is earlier than that of the source module file having the same file name, that source module file and all source module files specified after that source module file are assembled unconditionally.

If the order in which source module files are specified is changed, or a source module file is added or deleted, a source module file changed after the latest assembly, and all source module files that follow it will be assembled unconditionally.

To make the most of the assembly time reduction function, place debugged source module files before those which are currently being debugged.

Figure 4-1. Processing Flow of the Assembly Time Reduction Function (1/2)

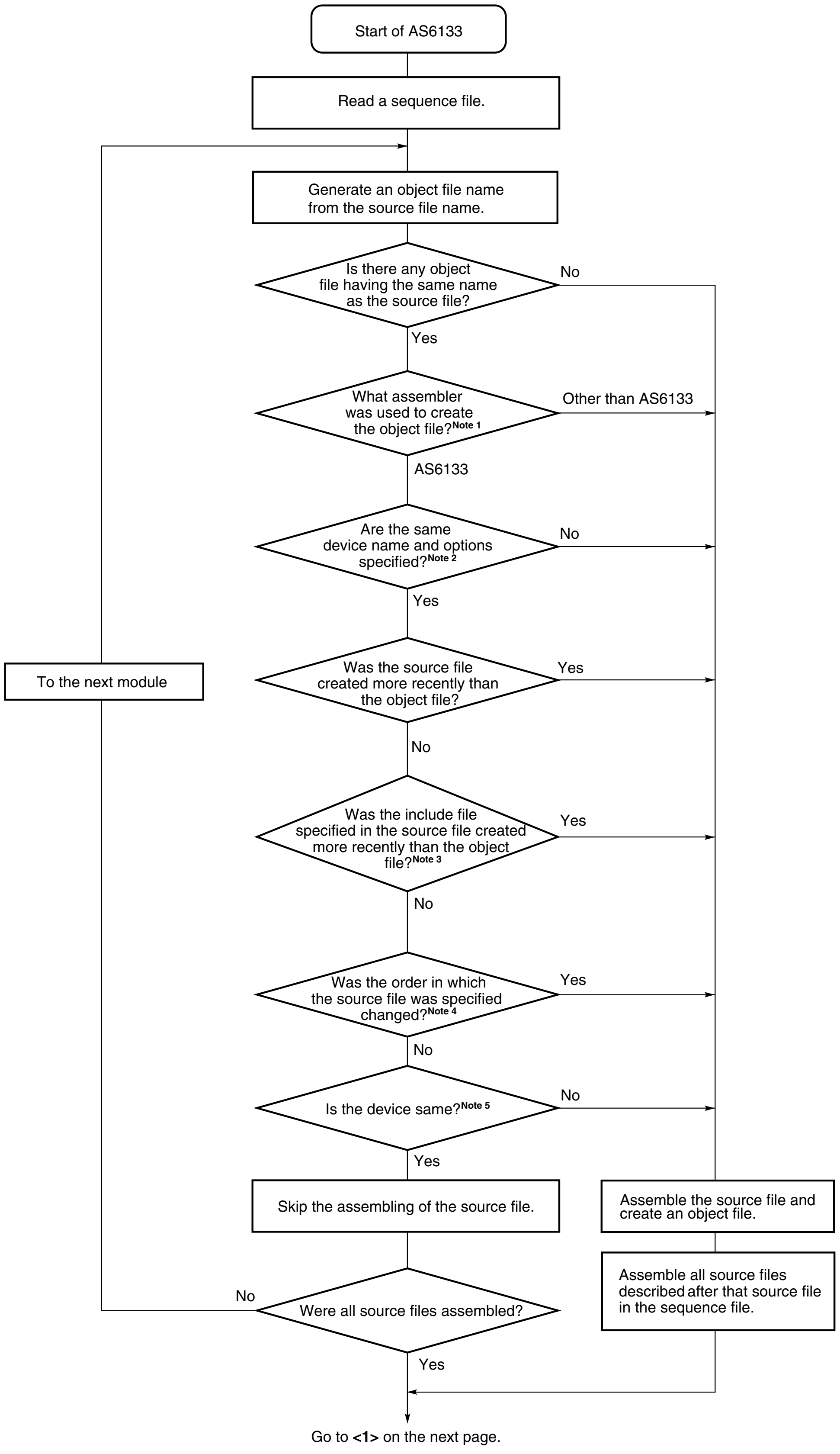
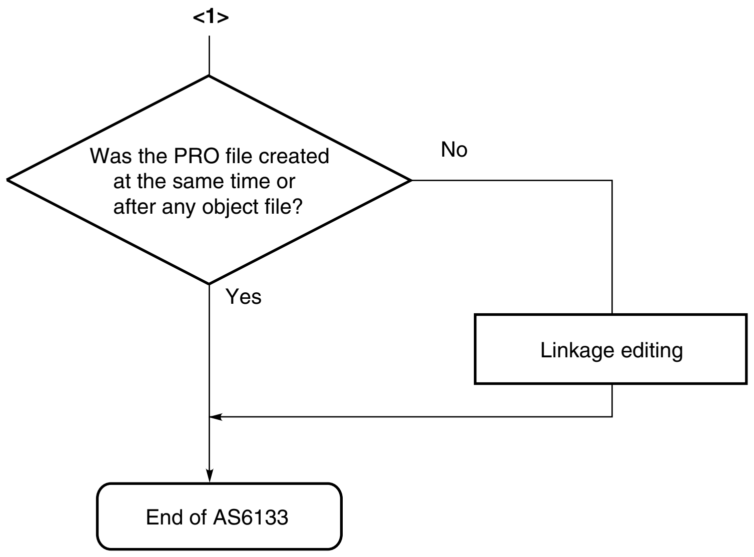

Start of AS6133
Read a sequence file.
Generate an object file name from the source file name.
Is there any object file having the same name as the source file?
No
Yes
What assembler was used to create the object file?Note 1
Other than AS6133
AS6133
Are the same device name and options specified?Note 2
No
Yes
To the next module
Was the source file created more recently than the object file?
Yes
No
Was the include file specified in the source file created more recently than the object file?Note 3
Yes
No
Was the order in which the source file was specified changed?Note 4
Yes
No
Is the device same?Note 5
No
Yes
Skip the assembling of the source file.
Assemble the source file and create an object file.
Assemble all source files described after that source file in the sequence file.
No
Were all source files assembled?
Yes
Go to <1> on the next page.

Figure 4-1. Processing Flow of the Assembly Time Reduction Function (2/2)


<1>
Was the PRO file created at the same time or after any object file?
No
Yes
Linkage editing
End of AS6133

Notes 1. The object file created using AS6133 begins with the "AS61" string.

2. The device name and options specified in the sequence file are checked with those specified in the object file.

3. The name of the include file is acquired from the object file.

4. The name of the immediately preceding source file is acquired from the object file.

5. The device information is acquired from the object file.

4.3.5 Assembly list file output function

An assembly list file can be output after assembly. An assembly option controls whether to output an assembly list file.

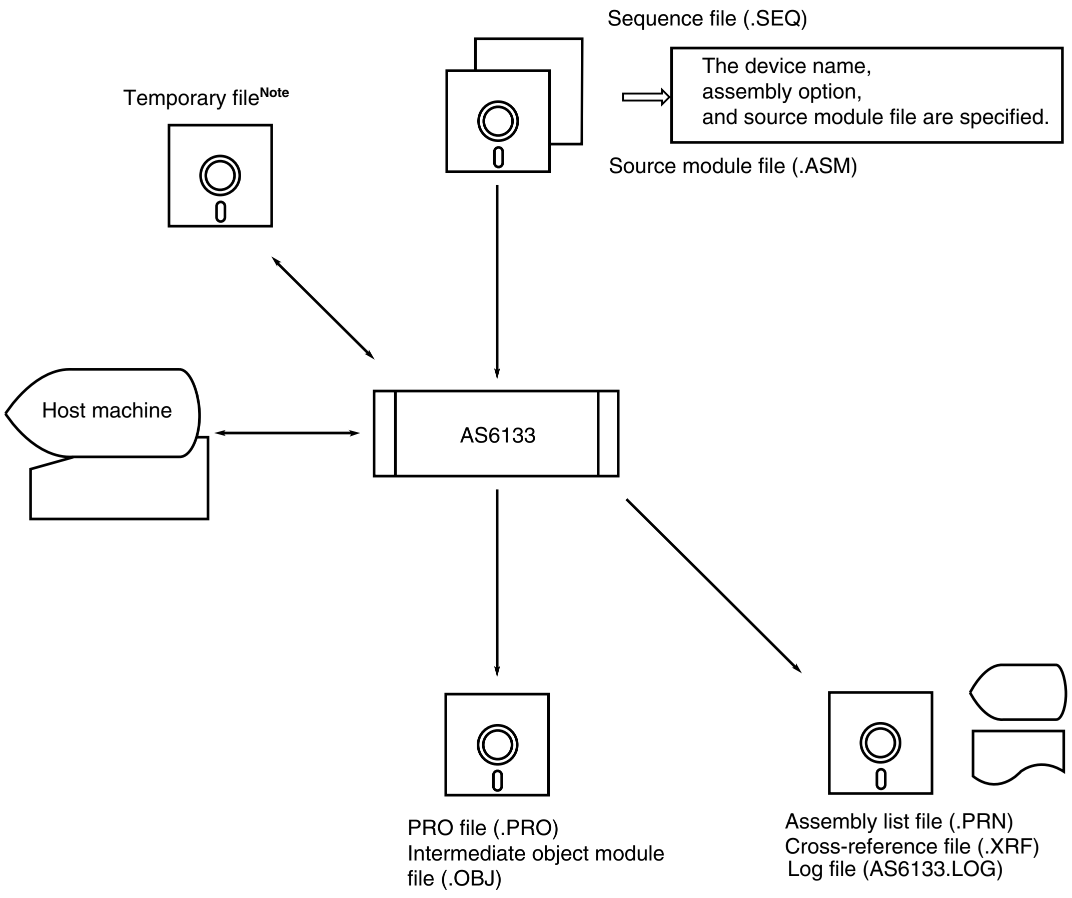
See **Chapter 5** for details.

4.3.6 Cross-reference list file output function

AS6133 creates a cross-reference list file.

See **Chapter 5** for details.

Figure 4-2. AS6133 Input/Output File Configuration


Sequence file (.SEQ)
The device name,
assembly option,
and source module file are specified.
Source module file (.ASM)
Temporary file[Note]
Host machine
AS6133
PRO file (.PRO)
Intermediate object module
file (.OBJ)
Assembly list file (.PRN)
Cross-reference file (.XRF)
Log file (AS6133.LOG)

Note The temporary file will be deleted at the end of assembly.

4.4 Assembler Start-Up Procedure

4.4.1 Input files needed when the assembler starts

The following files are necessary to start the assembler.

(1) Sequence file (.SEQ)

This file holds a device name, assembly options, and a source program file name that are required during assembly.

(2) Source module file (.ASM)

This file contains a source program.

See **Section 4.2** for details.

4.4.2 Starting the assembler

This section describes the actual procedure to start the assembler.
The assembler can be started by either of the following procedures.

Input methods

(1)
```
×>[directory]AS6133 ↵
```

(2)
```
×>[directory]AS6133Δ<sequence-file-name> ↵
```

×: current drive name

Cautions 1. To omit [directory] of AS6133, it is necessary to specify the PATH environment variable.
2. The sequence file and source file must be in the same directory.

The operation of the assembler is described below for the above two input methods separately.

(1) Starting by ×>[directory]AS6133 ↵

Insert the assembler disk into drive A, and the disk holding the sequence and source files in drive B.
Change the prompt to drive B, where the disk holding the sequence and source files is inserted, and enter as follows: "A:AS6133"

```
B>A:AS6133 ↵
```

The assembler will be loaded into memory and started to run.
After started, the assembler searches the current directory for the sequence file (.SEQ) as follows:

1. If there is one sequence file in the current directory
   The sequence file is read automatically, and assembly is carried out according to the contents of the sequence file.
2. If there is more than one sequence file in the current directory
   All sequence file names are numbered sequentially starting at 1, and listed on the display screen. The user should select the sequence file to be subjected to assembly.
3. If there is no sequence file to be selected
   The assembler stops running. Re-set the entry.

[Example]

(a) Starting the assembler under MS-DOS

```
B>A:AS6133 ↵
µPD6133 SERIES ASSEMBLER Vx.xx [xx xxx xx]
        Copyright (c) NEC Corporation 1995, 2000
=== SEQ FILE LIST IN CURRENT DIRECTORY ===

1) TEST1.SEQ  2) TEST.SEQ  3) TEST2.SEQ  4) TEST3.SEQ

Enter the sequence file number: 2 ↵
TEST.ASM assembly started on:  HH:MM:SS  MM/DD/YY
```

(b) Starting the assembler under PC DOS

```
B>A:AS6133 ↵
µPD6133 SERIES ASSEMBLER Vx.xx [xx xxx xx]
        Copyright (c) NEC Corporation 1995, 2000

=== SEQ FILE LIST IN CURRENT DIRECTORY ===

1) TEST1.SEQ  2) TEST.SEQ  3) TEST2.SEQ  4) TEST3.SEQ

SEQ FILE ? (SELECT NUMBER) = 2 ↵
TEST.ASM << ASSEMBLY START >>  HH:MM:SS  MM/DD/YY
```

(2) Starting the assembler by **×>[directory]AS6133Δ<sequence-file-name> ↵**

Insert the assembler disk into drive A, and the sequence and source file disk into drive B.
Enter "AS6133ΔB:SAMPLE.SEQ" in response to the prompt (A>).

```
A>AS6133ΔB:SAMPLE.SEQ ↵
```

This entry causes the assembler to be loaded into memory and to run according to the SAMPLE.SEQ sequence file in drive B.
The ".SEQ" extension can be left out from the sequence file name. If it is left out, it is assigned automatically.
If the specified sequence file is missing, the assembler ends running. Enter the correct sequence file name.

4.4.3 Aborting assembly

To abort the assembler, enter control+C (^C) from the console. On receiving ^C, the assembler closes all files and stops running.

After the assembler stops, the MS-DOS prompt (A>) appears.

4.5 Assembly Options

Assembly options are used to specify the files to be output during assembly, their types, related variables, and work drive.

Assembly options are specified when they are written in the sequence file. See **Section 4.4** for details.

If no assembly option is specified, the default assembly options (previously specified in the assembler) are used.

★ **Table 4-1. Assembly Options**

Option	Default[Note]	Description	Reference
HOS[T] NOH[OST]	HOST	Controls EB-6133 emulator output.	p.89
OBJ[=<directory>] NOO[BJ]	OBJ (disabled)	Controls object output.	p.90
PRO[=file-name[.PRO]] NOPRO	PRO (disabled)	Controls load module output.	p.91
LIS[T][=file-name[.PRN]] NOL[IST]	LIST (disabled)	Controls assembly list output.	p.92
XREF[=file-name[.XRF]] NOX[REF]	XREF (NOX)	Controls cross-reference list output.	p.93
ROW[=n]	ROW = 66 (enabled)	Specifies the number of lines to be output on one page of list output (50 to 250).	p.94
COL[UMN][=n]	COL = 80 (col = 132)	Specifies the number of columns to be output on one line of list output (72 to 256).	p.94
SEQ NOS[EQ]	SEQ (NOS)	Controls option information output.	p.95
TAB NOT[AB][=n]	NOTAB = 8 (enabled)	Controls tabs (1 to 255).	p.95
FOR[M] NOF[ORM]	FORM (enabled)	Controls form feed.	p.96
ZZZn = m	ZZZn = 0 (enabled)	Controls assembly variables.	p.96
WOR[K] = drive-name:	Current drive (enabled)	Specifies a work drive.	p.97
HEAD NOHEAD	HEAD (HEAD)	Controls list header output.	p.97
HEL[P]	–	Displays help messages.	p.98

Note Information enclosed in parentheses corresponds to the setting used when /HOST is specified. "Disabled" means that the default value is fixed. Only the currently "enabled" value can be used.

★ **Caution To use the EB-6133 emulator, it is always necessary to specify the HOST option.**

★

4.5.1 Option to control EB-6133 emulator information output

[Format]

$$\left[\left\{\begin{array}{l}\text{HOS[T]}\\\text{NOH[OST]}\end{array}\right\}\right]$$

Default value.../HOST

[Function]

This option specifies whether to output information necessary to use the EB-6133 emulator (μPD6133 Series development tool).

[Description]

(1) HOS[T]

The information about the EB-6133 emulator is output to the object file.

The following assembly options are specified forcibly:

/OBJ/PRO/LIST/NOXREF/COL = 132/NOSEQ

All the files related to the above assembly options are output to the directory where the sequence file is.

Caution When /HOST is selected, all related input files (source files) must be in the same directory as the sequence file.

(2) NOH[OST]

The information about the EB-6133 emulator is not output.

Caution If no option is specified, /HOST is specified as default assumption.

4.5.2 Option to control object file output

[Format]

$$\left[\begin{Bmatrix}\text{OBJ[=<directory>]}\\ \text{NOO[BJ]}\end{Bmatrix}\right]$$

Default value.../OBJ
When /HOST is specified...disabled

[Function]

This option specifies whether to output an intermediate object file. If the option specifies to output an intermediate object file, it also specifies the directory to which the file is to be output.

If the specified directory contains an intermediate object file having the same name as the source module file, and its creation date/time is more recent than the source module file, assembly will not be carried out.

[Description]

(1) /OBJ[=<directory>]
An intermediate object file will be output.

(2) /NOO[BJ]
No intermediate object file will be output.

(3) The option can specify only the directory to which an intermediate object file is to be output. It cannot specify the name of the intermediate object file.

(4) If no intermediate object file is to be output (/NOO is specified), the /PRO option is disabled.

★ (5) This option is disabled, if the /HOST option (EB-6133 emulator information output) is specified. In this case, the intermediate object file is always output to the directory where the sequence file is.

4.5.3 Option to control load module file (PRO file) output

[Format]

$$\left[\begin{Bmatrix}\text{PRO[=file-name[.PRO]]}\\ \text{NOP[RO]}\end{Bmatrix}\right]$$

Default value.../PRO
When /HOST is specified...disabled

[Function]

This option specifies whether to output a load module file (PRO file). If the option specifies to output a load module file, it also specifies the name of the load module file.

[Description]

(1) PRO[=file-name]
A PRO file is output.

- **Specifying no file name**
  A load module file is output to the directory where the sequence file is, and named after the sequence file, that is: sequence-file-name.PRO

- **Specifying a file name**
  A load module file is created under the specified file name. The file names that can be used include: AUX, CON, PRN, and NUL. These files are directed to the following devices.
  - AUX: RS-232C
  - CON: Console (usually CRT)
  - PRN: Printer
  - NUL: No file output

  The file name must be specified in format: [drive-name:[\directory\]]file-name
  If a file extension is omitted, ".PRO" is used.

(2) NOP[RO]
No load module file is output.

★ (3) This option is disabled, if the /HOST option (EB-6133 emulator information output) is specified. In this case, the PRO file is always output to the directory where the sequence file is.

4.5.4 Option to control assembly list file output

[Format]

```
[ { LIS[T][=file-name[.PRN]] } ]                    Default value.../LIST
[ { NOL[IST]                 } ]    When /HOST is specified...disabled
```

[Function]

This option specifies whether to output an assembly list file. If the option specifies to output an assembly list file, it also specifies the name of the assembly list file.

[Description]

(1) LIS[T]

An assembly list file is output.
The destination of output can be specified in either of the following two ways.

Specifying no file name

An assembly list file is created under the same name as the source file in the directory where the source file is. If a source program is split into several modules, an assembly list file corresponding to a specific source module file is created under the same name as that source module file in the directory where that source module file is. The file extension ".PRN" is used for the assembly list file.

Specifying a file name

An assembly list file is created under the specified file name. The file names that can be used include: AUX, CON, PRN, and NUL
The file name must be specified in format: [drive-name:[\directory\]]file-name
If a file extension is omitted, ".PRN" is used.

(2) NOL[IST]

No assembly list file is output.

★ (3) This option is disabled, if the /HOST option (EB-6133 emulator information output) is specified. In this case, the assembly list file is always output under the same name as the source file to the directory where the sequence file is. The file extension ".PRN" is used.

4.5.5 Option to control cross-reference list file output

[Format]

```
⎡⎧ XRE[F][=file-name[.XRF]] ⎫⎤                         Default value...XREF
⎣⎩ NOX[REF]                 ⎭⎦            When /HOST is specified.../NOX
```

[Function]

This option specifies whether to output a cross-reference file. If the option specifies to output a cross-reference file, it also specifies the name of the cross-reference file.

If the output of a cross-reference file is specified, a cross-reference file is output for each source module file on a one-to-one basis.

[Description]

(1) XRE[F]

A cross-reference file is output.

The destination of output can be specified in either of the following two ways.

Specifying no file name

<1> If an assembly list is output, the cross-reference list is output to the same file as the assembly list. In this case, the file name specified here must be the same as the assembly list file.

<2> If no assembly list is output, that is if NOL is specified, a cross-reference file is created under the same name as the source file in the directory where the source file is. In this case, the file extension ".XRF" is used for the cross-reference file.

Specifying a file name

A cross-reference file is created under the specified file name. This method is used to specify that the cross-reference list be output to a file different from the assembly list file. The file names that can be used include: AUX, CON, PRN, and NUL

The file name must be specified in format: [drive-name:[\directory\]]file-name

If a file extension is omitted, ".XRF" is used.

(2) NOX[REF]

No cross-reference file is output.

4.5.6 Option to control the number of lines to be output on one list output page (ROW NO.)

[Format]

[ROW=n]	Default value.../ROW = 66 When /HOST is specified...enabled

[Function]

This option specifies the number of lines per page in all list files (such as assembly list and cross-reference list files).

[Description]

"n" is the number of lines per page. It is a decimal number, and can range between 50 and 250 (inclusive).

4.5.7 Option to control the number of columns to be output on one list output line

[Format]

[COL[UMN]=n]	Default value.../COL = 80 When /HOST is specified.../COL = 132

[Function]

This option specifies the number of columns per line in all list files (such as assembly list, memory map, and cross-reference list files).

[Description]

"n" is the number of lines per page. It is a decimal number, and can range between 72 and 255 (inclusive).

4.5.8 Option to control option information output

[Format]

```
[ { SEQ       } ]                                   Default value.../SEQ
[ { NOS[EQ]   } ]                     When /HOST is specified.../NOS
```

[Function]

This option specifies whether to output the following information to the first page of the assembly list of each source module.

- Sequence file name specified when the assembler is started, and the contents of the sequence file (SEQ=)

[Description]

(1) SEQ
Information (described under [Function]) about the options is output to the first page of the assembly list file.

(2) NOSEQ
Information about the options is not output to the first page of the assembly list file. This information cannot be output separately from the assembly list.

(3) This option is disabled if /NOLIST is specified as the assembly list file output control option.

4.5.9 Tab control option

[Format]

```
[ { TAB              } ]                            Default value...NOT = 8
[ { NOTAB[=n]        } ]              When /HOST is specified...enabled
```

[Function]

This option specifies whether to use tab characters in the assembly list.

[Description]

(1) TAB
Tab characters are used in the assembly list. If this is selected, assembly is speeded, and the memory capacity required to store the files becomes smaller.

(2) NOT[AB]
No tab characters are used in the assembly list. A tab character (if there is one) is replaced with space characters so that the character next to the tab character is at the column that is a multiple of n (as counted from the beginning of the line). "n" is a decimal number, and can range between 1 and 255 (inclusive). If "n" is out of this range, an error is detected, and the assembler is aborted.
This option should be used for a printer that cannot recognize the tab character.

4.5.10 Form feed control option

[Format]

$\left[\left\{\begin{array}{l}\text{FOR[M]}\\ \text{NOF[ORM]}\end{array}\right\}\right]$	Default value.../FOR When /HOST is specified...enabled

[Function]

This option specifies whether the form of the output list be fed by a form feed character (0CH in 8-bit JIS code) or sets of CR/LF characters.

[Description]

(1) FOR[M]
The form of the output list is fed by a form feed character.

(2) NOF[ORM]
The form of the output list is fed by outputting CR/LF character sets repeatedly until the value specified in the ROW option (option to control the number of lines per output list page) is reached.

(3) This option should be used for a printer that cannot recognize the form feed character.
If FOR[M] is selected, assembly is speeded, and the memory capacity required to store the files becomes smaller.

4.5.11 Option to control assembly-time variables

[Format]

[ZZZn = m]	$0 \le n \le 9$ $0H \le m \le 0FFFFH$	Default value...ZZZn = 0 When /HOST is specified...enabled

[Function]

This option initializes the ZZZn assembly-time variable to the value m.

[Description]

(1) The evaluated value of m must fall in a range between 0H and 0FFFFH. If it is greater than 0FFFFH, it is assumed to be 0.

(2) m can be a binary, octal, decimal, or hexadecimal number. If a character string is specified as m, an error (invalid option) is reported, and the assembler is aborted.

(3) If the option is not specified when the assembly is started, the assembly-time variables are initially set to 0. This value remains effective until it is changed by a SET pseudo instruction.

4.5.12 Option to control a work drive

[Format]

[WOR[K] = drive-name:]	Default value...current drive When /HOST is specified...enabled

[Function]

This option specifies the name of a drive in which assembly work files are prepared.

[Description]

(1) Drive name specification
Only one drive name can be specified.

Example: WORK = A:

(2) All work files are deleted at the end of assembly.

4.5.13 Option to control list header output

[Format]

HEAD NOHEAD	Default value...HEAD When /HOST is specified...HEAD

[Function]

This option specifies whether to output the headers of lists such as an assembly list and cross-reference file list.

[Description]

(1) /HEAD
The header is output to each page of the list.

(2) /NOHEAD
The header is output only to the first page of the list. It is not output to the other pages.

(3) This option is applicable to the following lists.
- Assembly list file
- Cross-reference list file

4.5.14 Help message display

[Format]

```
HEL[P]
```

[Function]

This option displays the description of AS6133.

[Description]

This option cannot be specified in the sequence file. It can be specified only in format: AS6133Δ/HEL[P]

CHAPTER 5 ASSEMBLY OUTPUT LISTS

5.1 Types of Assembly Output Lists

AS6133 can output the following lists after assembly.

Table 5-1. Output Lists

Output File	Output File Extension	Assembly Option	Whether the List Is Output When/HOST Is Specified
Object file	.OBJ	/OBJ	○
PRO file	.PRO	/PRO	○
Assembly list	.PRN	/LIS[T]	○
Option information list	.PRN	/SEQ	
Cross-reference list	.XRF or .PRN	/XRE[F]	
Log file	AS6133.LOG		○

To output a list mentioned in Table 5-1, specify the corresponding assembly option when starting the assembler. See **Section 4.5** for how to specify it.

If it is unnecessary to output a list, prefix the corresponding assembly option with "NO" as in /NOLIST or /NOSEQ.

5.2 Controlling Each List Output Format

(1) Number of lines per page

The number of lines per page is determined by the ROW = n assembly option (where $50 \leq n \leq 250$). n is defaulted to 66.

(2) Number of columns per line

The number of columns per line is determined by the COL = n assembly option (where $72 \leq n \leq 255$). If the specified list output exceeds this value, the excess portion is cut out from listing. If a full-size character falls in the cut position, the cut position is shifted one place backward. n is defaulted to 80. If /HOST is specified, however, n is fixed at 132.

(3) Form feed control

The form is fed according to the FORM/NOFORM assembly option.

FORM................ A new page is selected when the FF character is detected (default).

NOFORM........... The form is advanced by outputting CR/LF characters repeatedly until the form is advanced as many lines as specified in the ROW assembly option.

Note FF (form feed) character .... 0CH in the 8-bit JIS code
LF (line feed) character .... 0AH in the 8-bit JIS code
CR (carriage return) character........... 0DH in the 8-bit JIS code

(4) Tab control

Tab control is carried out according to the TAB/NOTAB assembly option.

NOTAB = n.............. A tab character is replaced with space characters so that the character next to the tab character is at the column whose number (counted from the beginning of the line) equals a multiple of n (n is defaulted to 8).

TAB .... Tab characters are output.

5.3 Header Output

Lists other than a document list have a header (printed at the top of each page) consisting of the following information:

(1) Assembler name and version
(2) Device name
(3) Listing title
(4) Assembly date/time and page (module sequence number - page number within the module)
(5) Module name
Example: UPD6133.ASM

An assembly option can specify whether to output the header.
/HEAD (default) specifies that the header be printed at the top of each assembly list page.
/NOHEAD specifies that the header be output at the top of the first page only.

5.4 Assembler's Check Functions

The assembler checks each instruction in a source program to minimize errors that may occur when its object program is executed.

5.4.1 Error check for instructions exceeding the allowable number of bits

The assembler outputs a message if an instruction in a source program exceeds the allowable number of bits.

★ **(1) Instructions specifying immediate data**

ANL	A,	#data	: Error if the number of bits is greater than 4.
ORL	A,	#data	: Error if the number of bits is greater than 4.
XRL	A,	#data	: Error if the number of bits is greater than 4.
OUT	Pp,	#data	: Error if the number of bits is greater than 10. If bit 8 or 9 is 1, a warning message is output, and bits 8 and 9 are reset to 0.
MOV	A,	#data	: Error if the number of bits is greater than 4.
MOV	Rr,	#data	: Error if the number of bits is greater than 10. If bit 8 or 9 is 1, a warning message is output, and bits 8 and 9 are reset to 0.
MOV	T,	#data	: Error if the number of bits is greater than 10.
MOV	M0,	#data	: Error if the number of bits is greater than 10.
MOV	M1,	#data	: Error if the number of bits is greater than 10.
STTS		#data	: Error if the number of bits is greater than 4.

(2) DT and DW instructions

DT instruction	: Error if the number of bits is greater than 10.
DW instruction	: Error if the number of bits is greater than 10.
DW instruction	: If bit 8 or 9 is 1, a warning message is output, and bits 8 and 9 are reset to 0.

5.4.2 Check to prevent a program crash

If the supply voltage fluctuates during operation, or a power-on reset fails to take place, the program counter may become undefined, possibly resulting in a program crash. If the program counter points to an address at which there is no programmed instruction, and the code at that address happens to match the operation code of a jump or HALT instruction, an endless loop may result.

To prevent a program crash, if the object code pointed to by the program counter happens to match the operation code of a branch or HALT instruction, the assembler outputs a warning message and displays an instruction that will be generated. The instructions generated in this case include: JMP, JC, JNC, JF, JNF, CALL, RET, and HALT.

If a warning message is output, check the instruction that will be generated. If the instruction can cause an endless loop, the program should be corrected.

★ **5.4.3 Check for the destination of a branch instruction (automatic check on BANK0 and BANK1)**

This check is made for a device in which the number of ROM words is greater than 1,024.

To branch execution without the BANK number at the branch destination described in the mnemonic, describe as follows:

- Object code of J×0 or CALL0 to branch to BANK0 (0 to 1,023 instructions)
- Object code of J×1 or CALL1 to branch to BANK1 (1,024 to 2,047 instructions)
- Object code of J×2 or CALL2 to branch to BANK2 (2,048 to 3,071 instructions)
- Object code of J×3 or CALL3 to branch to BANK3 (3,072 instructions or more)

If J×0, J×1, J×2, J×3, CALL0, CALL1, CALL2, or CALL3 is described with a BANK number, an error occurs.

- **Branch instruction**

		Source instruction	
JMP0	addr	JMP	addr
JMP1	addr		
JMP2	addr		
JMP3	addr		
JC0	addr	JC	addr
JC1	addr		
JC2	addr		
JC3	addr		
JNC0	addr	JNC	addr
JNC1	addr		
JNC2	addr		
JNC3	addr		
JF0	addr	JF	addr
JF1	addr		
JF2	addr		
JF3	addr		
JNF0	addr	JNF	addr
JNF1	addr		
JNF2	addr		
JNF3	addr		

- **Subroutine instruction**

		Source instruction	
CALL0	addr	CALL	addr
CALL1	addr		
CALL2	addr		
CALL3	addr		

5.4.4 Check for output to an input-only port

If an output instruction is coded for an input-only port, an error message is output.

- **Input-only port**

P11	(KI3 to KI0)
P01	(S1/LED,S0)

- **Output instruction**

```
OUT   P11, A
OUT   P01, A
OUT   P1, #data
```

⋆ 5.4.5 Check for input/output instructions for nonexisting ports

A warning message is output if an input or output instruction is coded for a nonexisting port.

Port / Device	P10 (KI/O7 to KI/O4)	P00 (KI/O0 to KI/O3)	P11 (KI3 to KI0)	P01 S1LED S0	P12 I/OPull I/OMode	P02 (I/O3 to I/O0)
D6133	○	○	○	○	×	×
D6134	○	○	○	○	×	×
D6135	○	○	○	○	○	○
D6603	○Note 1	○	○	○Note 2	×	×
D6604	○	○	○	○	×	×
D6605	○	○	○	○	○	○
D63	○	○	○	○	×	×
D63A	○	○	○	○	×	×
D64	○	○	○	○	×	×
D64A	○	○	○	○	×	×
D65	○	○	○	○	○	○
D62	○	○	○	○	×	×
D62A	○	○	○	○	×	×
D6132	○	○	○	○	×	×
D6132A	○	○	○	○	×	×
D67	○	○	○	○	×	×
D68	○	○	○	○	×	×
D69	○	○	○	○	×	×

○: No warning message is output. ×: A warning message is output.

Caution Refer to the device name of the supported mask ROM version device when using the μPD61P34B, 66P04B, 6P4B, 6P5, or 6P9.

Notes 1. D6603 does not have KI/O7 to KI/O5 but a warning message is not output.

2. D6603 does not have S0 but a warning message is not output.

CHAPTER 6 ERROR MESSAGES

6.1 Errors Detected at Start-Up and Run Time

If a parameter specified at start-up is incorrect, or an error occur at run time, AS6133 displays error messages, then stops prematurely.

Message text	file not found
Cause	A file specified at start-up is not found in a specified directory on a specified drive
System action	AS6133 stops running.
User response	Specify the correct file.
Message text	invalid option
Cause	A specified option is invalid (such as invalid option name or parameter).
System action	The invalid option is indicated, and assembly is aborted.
User response	Specify the correct option.
Message text	invalid option value
Cause	A value specified for an option is invalid (a value out of the describable range was specified).
System action	The invalid option is indicated, and assembly is aborted.
User response	Specify the correct option.
Message text	out of memory
Cause	The memory capacity is insufficient.
System action	Assembly is aborted.
User response	Decrease the number of options used, increase memory, or change the /WORK drive specification.

In the following case, a message is displayed, but assembly is not aborted.

Message text	HALT table overflow
Cause	The HALT area has overflowed.
System action	There are more than 32 HALT instructions, and information about the HALT instructions is not preserved.
User response	Decrease the number of the HALT instructions.

(1) Error message format

An error message includes a source statement in which the error occurred. The displayed source statement line is followed by a line containing the source file name, line number, error type, error number, and error message text. The error message ends with the numbers of errors and warnings.

[Example]

```
             INC     C                                     <——  Source statement in which the
                                                                error is detected
DIRTEST.ASM  (3):  S  error-number  058:  Undefined symbol  <——  Source file name, line number,
                                                                error message text
    <1>      <2>  <3>               <4>        <5>

Number of errors = 1    Number of warnings = 0             <——  Total numbers of errors and
                                                                warnings
```

<1>: Source file name

<2>: Line number

<3>: Error type

1:	ROM area overflow
1:	A location out of the program memory was referenced.
A:	Addressing error
F:	Include file open error
I:	Invalid operand value in the ORG pseudo instruction
I:	Invalid data strength
I:	Error related to an INCLUDE file
M:	MACRO management file I/O error
M:	Memory area overflow
M:	Invalid instruction
N:	Nest stack overflow
O:	Mask option definition block error
O:	Duplicate mask option specification
O:	Invalid operand
P:	Invalid pseudo instruction
P:	Invalid statement
R:	REPT area overflow
R:	A statement was written out of the program memory area.
R:	A reserved word was used where it should not be.
S:	Duplicate symbol definition or undefined symbol
S:	Symbol area overflow
S:	Syntax stack overflow
S:	Format or syntax error
T:	Invalid operand
U:	Undefined symbol
V:	Invalid operand value or count
W:	Warning

<4>: Error number
<5>: Error message text

Caution The warning messages described in Section 5.4.2 may not include a source statement in which an error is detected.

11	Code O	Message text	Illegal first operand type
		Cause	The first operand is invalid.
		User response	Correct the expression.
12	Code O	Message text	Illegal second operand type
		Cause	The second operand is invalid.
		User response	Correct the expression.
14	Code V	Message text	Illegal first operand value
		Cause	The first operand value is incorrect.
		User response	Make sure that the operand value is acceptable to the device model of interest.
20	Code W	Message text	Unreference symbol
		Cause	The symbol has not been referenced.
		User response	Check whether the symbol is necessary. If the symbol is unnecessary, delete it. If the symbol is necessary, reference it.
21	Code P	Message text	No IF directive
		Cause	An IF statement is missing.
		User response	Write an IF statement in the correct position.
25	Code S	Message text	Symbol define error
		Cause	The symbol definition is incorrect.
		User response	Correct the symbol define pseudo instruction and its operand.
27	Code P	Message text	No OPTION statement
		Cause	An OPTION statement is missing.
		User response	Add an OPTION statement, because an ENDOP statement was specified when an OPTION statement was not.
28	Code P	Message text	No END directive
		Cause	An END statement is missing.
		User response	Add an END statement.
29	Code P	Message text	No ENDIF directive
		Cause	An ENDIF is missing.
		User response	Write an ENDIF statement in the correct position.
31	Code P	Message text	No ENDR directive
		Cause	An ENDR statement is missing.
		User response	Write an ENDR statement in the correct position.

32	Code P	Message text	No ENDM directive
		Cause	MACRO has no corresponding ENDM statement.
		User response	Write an ENDM statement in the correct position.
33	Code P	Message text	No ENDP directive
		Cause	An ENDP statement is missing.
		User response	Write an ENDP statement in the correct position.
35	Code N	Message text	Nesting overflow
		Cause	The nest stack has overflowed.
		User response	Decrease the total nesting depth of IF and REPT-ENDR statements to or below level 40.
36	Code O	Message text	Operand count error
		Cause	An attempt was made to specify more operands than allowed.
		User response	Decrease the number of operands.
37	Code S	Message text	Syntax error
		Cause	There is a syntax error.
		User response	Correct the statement.
39	Code S	Message text	Symbol area overflow
		Cause	The symbol area has overflowed.
		User response	Decrease the number of symbols, or increase the size of the usable memory area.
41	Code P	Message text	Invalid ENDR statement
		Cause	The ENDR statement is invalid.
		User response	Write the ENDR statement in the correct position.
42	Code P	Message text	Invalid EXITR statement
		Cause	The EXITR statement is invalid.
		User response	Write the EXITR statement in the correct position.
43	Code P	Message text	Invalid ENDM statement
		Cause	The ENDM statement is invalid.
		User response	Write the ENDM statement in the correct position.
44	Code V	Message text	Invalid value
		Cause	There is an invalid value.
		User response	Correct the value.
45	Code T	Message text	Invalid operand
		Cause	There is an invalid operand.
		User response	Correct the operand.
47	Code R	Message text	Out of address range (3)
		Cause	A statement is written out of the program memory area (3).
		User response	Shift the statement into the program memory area.

49	Code R	Message text	Used reserved word
		Cause	A reserved word is used where it should not be.
		User response	Do not use a reserved word in the name of a symbol.
51	Code I	Message text	Invalid data length
		Cause	The data length is invalid.
		User response	Do not try to use more characters than allowed for the data.
52	Code N	Message text	Include nesting error
		Cause	There two many include nesting levels.
		User response	Decrease the number of include nesting levels to within 8.
53	Code O	Message text	Duplicated OPTION directive
		Cause	There are duplicate OPTION pseudo instruction definitions.
		User response	Do not write more than one option block in a source program.
55	Code R	Message text	Rept area overflow
		Cause	The REPT area has overflowed.
		User response	Decrease the number of repeat definition nesting levels to within 8.
57	Code S	Message text	Symbol multi defined
		Cause	There are duplicate symbol definitions.
		User response	Use different symbol names.
58	Code S	Message text	Undefined symbol
		Cause	There is an undefined symbol.
		User response	Write a defined symbol, or define one.
59	Code P	Message text	Invalid Pseudo
		Cause	There is an invalid pseudo instruction.
		User response	Correct the pseudo instruction.
61	Code F	Message text	Include file open error
		Cause	An include file cannot be opened.
		User response	Specify a correct include file, or expand the memory area.
62	Code S	Message text	Parser stack overflow
		Cause	The syntax stack area has overflowed.
		User response	Decrease the nesting depth of (and) pairs below level 17 and number of operators below 32.
65	Code W	Message text	Statement after END
		Cause	An END statement is followed by another statement.
		User response	Remove the statement after the END statement.
67	Code A	Message text	Address error
		Cause	A specified address is incorrect.
		User response	Specify an address that is acceptable to the device model of interest.

68	Code W	Message text	Operation in OPTION block
		Cause	A mask option definition block contains an instruction.
		User response	Remove the instruction.
71	Code O	Message text	Illegal first operand type and value
		Cause	The value of the first operand is invalid.
		User response	Correct the operand.
72	Code O	Message text	Illegal second operand type and value
		Cause	The value of the second operand is invalid.
		User response	Correct the operand.
74	Code U	Message text	Undefined first operand symbol
		Cause	The symbol in the first operand is undefined.
		User response	Use a defined symbol, or define the symbol already used in the operand.
75	Code U	Message text	Undefined second operand symbol
		Cause	The symbol in the second operand is undefined.
		User response	Use a defined symbol, or define the symbol already used in the second operand.
77	Code O	Message text	Not found Mask-option block
		Cause	A mask option definition block is missing.
		User response	Specify a mask option using an OPTION pseudo instruction.
85	Code P	Message text	Invalid ENDP statement
		Cause	There is an invalid ENDP statement.
		User response	Specify an ENDP statement that corresponds to PUBLIC BELOW.
98	Code W	Message text	Invalid instruction of last address in program
		Cause	The last instruction in the program is neither JMP nor RET.
		User response	If the last instruction is not DW or DT, specify either a JMP or RET instruction.
99	Code V	Message text	Over max value
		Cause	The specified value is greater than an allowable number of bits, file name [address].
		User response	Decrease the value to within an allowable number of bits. If a value is determined at linkage time, the file name and address are displayed.
100	Code W	Message text	Over effective value
		Cause	The specified value is greater than a valid number of bits.
		User response	Reset the invalid bits to 0. Decrease the value to within an allowable number of bits.
101	Code P	Message text	Output request for read only port
		Cause	An output instruction was specified for an input-only port.
		User response	Specify an output instruction only for an input/output or output-only port.

102	Code W	Message text	Input/Output request for non-existent port
		Cause	An input/output instruction was specified for a nonexisting port.
		User response	Specify the input/output instruction for an existing port.
103	Code W	Message text	Same operand value with branch or HALT
		Cause	An operand value happened to match the operation code of a branch or HALT instruction.
		User response	A value specified in the second or subsequent operand of an instruction or a value defined in a Define instruction happened to match the operation code of a branch or HALT instruction. For a branch instruction, the error message contains the branch address. For the HALT instruction, the error message contains the operand value. Check the operation of an instruction executed after the program counter becomes undefined.
104	Code S	Message text	The source file does not exist in the same directory with SEQ file
		Cause	When the /HOST option is specified, the source file is not in the same directory as the sequence file.
		User response	Move the source file to the same directory where the sequence file is.
105	Code I	Message text	Too many INCLUDE file
		Cause	There are too many INCLUDE files.
		User response	Decrease the number of INCLUDE files per source file to within 16.
106	Code I	Message text	Too long INCLUDE file name
		Cause	There are too many characters in the names of INCLUDE files.
		User response	Decrease the number of characters in the INCLUDE file names per source file to within 255.

★

APPENDIX A CONSTRAINTS

This appendix explains the constraints of AS6133 V2.21 or later.

No.	Constraint
1	When /HOST option is specified, a drive name and a directory name must not be included in the source file name described in the sequence file. Also when /NOH[OST] option is specified, a relative path must not be specified in specifying a source file name to be described in the sequence file. For the description format of a source file name, see **Section 3.2.4**.
2	The μPD6P4 cannot be used with program memory of 2,016 words.

★

APPENDIX B REVISION HISTORY

Here is the revision history of this manual. "Location" indicates the chapter of the edition.

Edition	Major Revision from Preceding Edition	Location
3rd edition	Deleting description "separate volume of SM6133" or "supplied with SM6133" because AS6133 assembler is separated from SM6133 simulator.	Throughout
	Changing supported debugger from NEC's SM6133 simulator to Naito Densei Machida Mfg's EB-6133	
	Adding μPD63 Series as target device	
	Adding series name and device that can be supported	PREFACE
	Changing description of PC-9800 series, IBM PC/AT compatibles, and assembler	
	Adding USEPOC/NOUSEPOC, USECAP/NOUSECAP	PART I LANGUAGE, CHAPTER 3 PSEUDO INSTRUCTIONS AND CONTROL INSTRUCTIONS
	Adding description on supported debugger	PART II OPERATION, CHAPTER 1 PRODUCT OVERVIEW
	Adding description on install	PART II OPERATION, CHAPTER 2 BEFORE EXECUTION
	Changing device name that can be described and description of supported device	PART II OPERATION, CHAPTER 3 SEQUENCE FILE
	Changing description in [Example] of assembler start-up procedure	PART II OPERATION, CHAPTER 4 ASSEMBLER FUNCTIONS
	Adding description on instructions specifying immediate data	PART II OPERATION, CHAPTER 5 ASSEMBLY OUTPUT LISTS
	Changing description of check for the destination of a branch instruction	
	Adding device to check for input/output instructions for nonexisting ports	
	Addition	APPENDIX A CONSTRAINTS

[MEMO]

NEC

Facsimile Message

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

From:

Name

Company

Tel. FAX

Address

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: +1-800-729-9288
+1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6462-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: +82-2-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: +886-2-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title: ______________________

Document number: ______________________ Page number: __________

____

____

____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❏	❏	❏	❏
Technical Accuracy	❏	❏	❏	❏
Organization	❏	❏	❏	❏

CS 01.2