

NEC

User's Manual

RA78K4 Ver. 1.50 or Later

Assembler Package

Language

Target Devices
78K/IV Series

Document No. U15255EJ1V0UM00 (1st edition)
Date Published September 2001 N CP(K)

© NEC Corporation 2001
Printed in Japan

[MEMO]

Windows and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

HP-UX is a trademark of Hewlett-Packard Company.

SunOS is a trademark of Sun Microsystems, Inc.

- **The information in this document is current as of July, 2001. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**

- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.

- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.

- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.

- NEC semiconductor products are classified into the following three quality grades: "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product before using it in a particular application.

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

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

"Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC semiconductor products is "Standard" unless otherwise expressly specified in NEC's data sheets or data books, etc. If customers wish to use NEC semiconductor products in applications not intended by NEC, they must contact an NEC sales representative in advance to determine NEC's willingness to support a given application.

(Note)

(1) "NEC" as used in this statement means NEC Corporation and also includes its majority-owned subsidiaries.

(2) "NEC semiconductor products" means any semiconductor product developed or manufactured by or for NEC (as defined above).

M8E 00.4

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

INTRODUCTION

This manual is designed to facilitate correct understanding of the basic functions of each program in the **RA78K4 Assembler Package** (hereafter called **RA78K4**) and the methods of describing source programs.

This manual does not cover how to operate the respective programs of the RA78K4. Therefore, after you have comprehended the contents of this manual, read the **RA78K4 Assembler Package User's Manual Operation (U15254E)** (hereafter called **Operation**) to operate each program in the assembler package.

Descriptions related to the RA78K4 in this manual apply to Ver. 1.50 or later.

[Target Readers]

This manual is intended for user engineers who understand the functions and instructions of the microcontroller (78K/IV Series) subject to development.

[Organization]

This manual consists of the following six chapters and appendices.

CHAPTER 1 GENERAL

Outlines all of the basic functions of the RA78K4.

CHAPTER 2 HOW TO DESCRIBE SOURCE PROGRAMS

Outlines how to describe source programs, and explains the operators of the assembler.

CHAPTER 3 DIRECTIVES

Explains how to write and use directives, including application examples.

CHAPTER 4 CONTROL INSTRUCTIONS

Explains how to write and use control instructions, including application examples.

CHAPTER 5 MACROS

Explains all macro functions, including macro definition, macro reference, and macro expansion. Macro directives are explained in **CHAPTER 3 DIRECTIVES**.

CHAPTER 6 PRODUCT UTILIZATION

Introduces some measures recommended for describing a source program.

APPENDICES

These contain a list of reserved words, a list of directives, the maximum performance, characteristics, and an index.

The instruction sets are not detailed in this manual. For these instructions, refer to the user's manual of the microcontroller for which software is being developed.

Also, for instructions on architecture, refer to the user's manual (hardware version) of each microcontroller for which software is being developed.

[How to Read This Manual]

Those using an assembler for the first time are encouraged to read from **CHAPTER 1 GENERAL** of this manual. Those who have a general knowledge of assembler programs may skip **CHAPTER 1 GENERAL** of this manual. However, be sure to read **1.2 Reminders Before Program Development** and **CHAPTER 2 HOW TO DESCRIBE SOURCE PROGRAMS**.

Those who wish to know the directives and control instructions of the assembler are encouraged to read **CHAPTER 3 DIRECTIVES** and **CHAPTER 4 CONTROL INSTRUCTIONS**, respectively. The format, function, use, and application examples of each directive or control instruction are detailed in these chapters.

[Conventions]

The following symbols and abbreviations are used throughout this manual.

- ∴: Same format is repeated.
- []: Characters enclosed in these brackets can be omitted.
- { }: One of the items in { } is selected.
- “ ”: Characters enclosed in “ ”(quotation marks) are a character string.
- ‘ ’: Characters enclosed in ‘ ’ (single quotation marks) are a character string.
- (): Characters between parentheses are a character string.
- < >: Characters (mainly title) enclosed in these brackets are a character string.
- : An underline is used to indicate an important point or, in an application example, an input character string.
- Δ: Indicates one or more blank characters or tabs.
- /: Character delimiter
- ~: Continuity
- Boldface**: Characters in boldface are used to indicate an important point or reference point.

[Related Documents]

The documents (user's manuals) related to this manual are listed below.

The related documents indicated in this publication may include preliminary versions.

However, preliminary versions are not marked as such.

Document Name		Document No.
RA78K4 Assembler Package	Operation	U15254E
	Language	This manual
	Structured Assembler Preprocessor	U11743E
CC78K4 C Compiler	Operation	To be prepared
	Language	To be prepared
SM78K4 System Simulator	Reference (Windows™ Based Operation)	U10093E
SM78K Series System Simulator V1.40 or later	External Part User Open Interface Specifications	U10092E
ID78K Series Integrated Debugger Ver.2.30 or later	Operation (Windows Based)	U15185E
ID78K4 Integrated Debugger	Reference (Windows Based Operation)	U10440E
78K/IV Series Real-Time OS RX78K/IV	Fundamental	U10603E
	Installation	U10604E

CONTENTS

CHAPTER 1 GENERAL	13
1.1 Assembler Overview	13
1.1.1 What is an assembler?	14
1.1.2 What is a relocatable assembler?.....	16
1.2 Reminders Before Program Development	18
1.2.1 Maximum performance characteristics of RA78K4	18
1.3 Features of RA78K4	20
CHAPTER 2 HOW TO DESCRIBE SOURCE PROGRAMS	21
2.1 Basic Configuration of Source Program	21
2.1.1 Module header	22
2.1.2 Module body	23
2.1.3 Module tail.....	24
2.1.4 Overall configuration of source program.....	24
2.1.5 Description example of source program	25
2.2 Description Format of Source Program	28
2.2.1 Configuration of statements	28
2.2.2 Character set	29
2.2.3 Fields that make up a statement.....	32
2.3 Expressions and Operators	44
2.3.1 Functions of operators	45
2.3.2 Restrictions on operations	61
2.4 Bit Position Specifier	67
2.5 Characteristics of Operands	70
2.5.1 Size and address range of operand value	70
2.5.2 Size of operands required for instructions	73
2.5.3 Symbol attributes and relocation attributes of operands.....	75
CHAPTER 3 DIRECTIVES	80
3.1 Overview of Directives	80
3.2 Segment Definition Directives	81
(1) CSEG (code segment)	83
(2) DSEG (data segment)	87
(3) BSEG (bit segment)	91
(4) ORG (origin).....	96
3.3 Symbol Definition Directives	99
(1) EQU (equate)	100
(2) SET (set)	104
3.4 Memory Initialization and Area Reservation Directives	106
(1) DB (define byte)	107
(2) DW (define word).....	109
(3) DG (dg).....	111
(4) DS (define storage)	113
(5) DBIT (define bit)	115
3.5 Linkage Directives	116

(1) EXTRN (external)	117
(2) EXTBIT (external bit)	119
(3) PUBLIC (public)	121
3.6 Object Module Name Declaration Directive	123
(1) NAME (name)	124
3.7 Automatic Branch Instruction Selection Directive.....	125
(1) BR (branch)	126
(2) CALL (call).....	128
3.8 General-Purpose Register Selection Directive.....	130
(1) RSS (register set select)	131
3.9 Macro Directives	134
(1) MACRO (macro).....	135
(2) LOCAL (local).....	137
(3) REPT (repeat).....	140
(4) IRP (indefinite repeat).....	142
(5) EXITM (exit from macro).....	144
(6) ENDM (end macro).....	147
3.10 Assembly Termination Directive	149
(1) END (end)	150
CHAPTER 4 CONTROL INSTRUCTIONS	151
4.1 Overview of Control Instructions	151
4.2 Processor Type Specification Control Instruction.....	152
(1) PROCESSOR (processor)	153
4.3 Debug Information Output Control Instructions	154
(1) DEBUG/NODEBUG (debug/nodebug)	155
(2) DEBUGA/NODEBUGA (debuga/nodebuga)	156
4.4 Cross-Reference List Output Specification Control Instructions.....	157
(1) XREF/NOXREF (xref/noxref)	158
(2) SYMLIST/NOSYMLIST (symlist/nosymlist).....	159
4.5 Inclusion Control Instruction	160
(1) INCLUDE (include).....	161
4.6 Assembly List Control Instructions	164
(1) EJECT (eject).....	165
(2) LIST/NOLIST (list/nolist).....	167
(3) GEN/NOGEN (generate/no generate)	169
(4) COND/NOCOND (condition/no condition)	171
(5) TITLE (title).....	173
(6) SUBTITLE (subtitle).....	176
(7) FORMFEED/NOFORMFEED (formfeed/noformfeed).....	179
(8) WIDTH (width)	180
(9) LENGTH (length).....	181
(10) TAB (tab).....	182
4.7 Conditional Assembly Control Instructions	183
(1) IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF	184
(2) SET/RESET (set/reset).....	188
4.8 SFR Area Change Control Instructions.....	190
(1) CHGSFR/CHGSFRA (change sfr area/change sfr area).....	191

4.9 Other Control Instructions	192
CHAPTER 5 MACROS	193
5.1 Overview of Macros	193
5.2 Utilization of Macros	194
5.2.1 Macro definition.....	194
5.2.2 Macro reference	195
5.2.3 Macro expansion.....	196
5.3 Symbols Within Macros.....	197
5.4 Macro Operators.....	200
CHAPTER 6 PRODUCT UTILIZATION	202
APPENDIX A LIST OF RESERVED WORDS	204
APPENDIX B LIST OF DIRECTIVES	205
APPENDIX C MAXIMUM PERFORMANCE CHARACTERISTICS.....	207
APPENDIX D INDEX.....	208

LIST OF FIGURES

Figure No.	Title	Page
1-1	RA78K4 Assembler Package.....	13
1-2	Flow of Assembler.....	14
1-3	Development Process of Products Employing Microcontrollers.....	15
1-4	Reassembly for Debugging.....	17
1-5	Program Development Using Existing Module.....	17
2-1	Configuration of Source Module.....	21
2-2	Overall Configuration of Source Module	24
2-3	Examples of Source Module Configurations	24
2-4	Configuration of Sample Program.....	25
2-5	Fields That Make Up a Statement.....	28
3-1	Memory Location of Segments	82
3-2	Relocation of Code Segment	83
3-3	Relocation of Data Segment	87
3-4	Relocation of Bit Segment	91
3-5	Location of Absolute Segment.....	96
3-6	Relationship of Symbols Between Two Modules	116

LIST OF TABLES

Table No.	Title	Page
1-1	Maximum Performance Characteristics of Assembler.....	18
1-2	Maximum Performance Characteristics of Linker.....	19
2-1	Instructions That Can Be Described in Module Header	22
2-2	Symbol Types.....	32
2-3	Names of Segments Automatically Generated by Assembler.....	34
2-4	Symbol Attributes and Values	35
2-5	Methods of Representing Numeric Constants.....	38
2-6	Special Characters That Can Be Described in Operand Field	40
2-7	Types of Operators.....	44
2-8	Order of Precedence of Operators	45
2-9	Types of Relocation Attributes	61
2-10	Combinations of Terms and Operators by Relocation Attribute	62
2-11	Combinations of Terms and Operators by Relocation Attribute (External Reference Terms)	64
2-12	Types of Symbol Attributes in Operations	65
2-13	Combinations of Terms and Operators by Symbol Attribute	66
2-14	Combinations of 1st and 2nd Terms by Relocation Attribute.....	69
2-15	Values of Bit Symbols.....	69
2-16	Ranges of Operand Values of Instructions.....	71
2-17	Ranges of Operand Values of Directives.....	72
2-18	Attributes of Instruction Operands	76
2-19	Properties of Described Symbols as Operands of Directives	78
3-1	List of Directives.....	80
3-2	Segment Definition Methods and Memory Address Location	81
3-3	Relocation Attributes of CSEG	84
3-4	Default Segment Names of CSEG	85
3-5	Relocation Attributes of DSEG	88
3-6	Default Segment Names of DSEG	89
3-7	Relocation Attributes of BSEG	92
3-8	Default Segment Names of BSEG	94
3-9	Representation Formats of Operands Indicating Bit Values	101
3-10	Absolute Names and Function Names of General-Purpose Registers.....	130
4-1	List of Control Instructions.....	151
4-2	Control Instructions and Assembler Options	152

CHAPTER 1 GENERAL

This chapter describes the role of the RA78K4 in microcontroller software development and the features of the RA78K4.

1.1 Assembler Overview

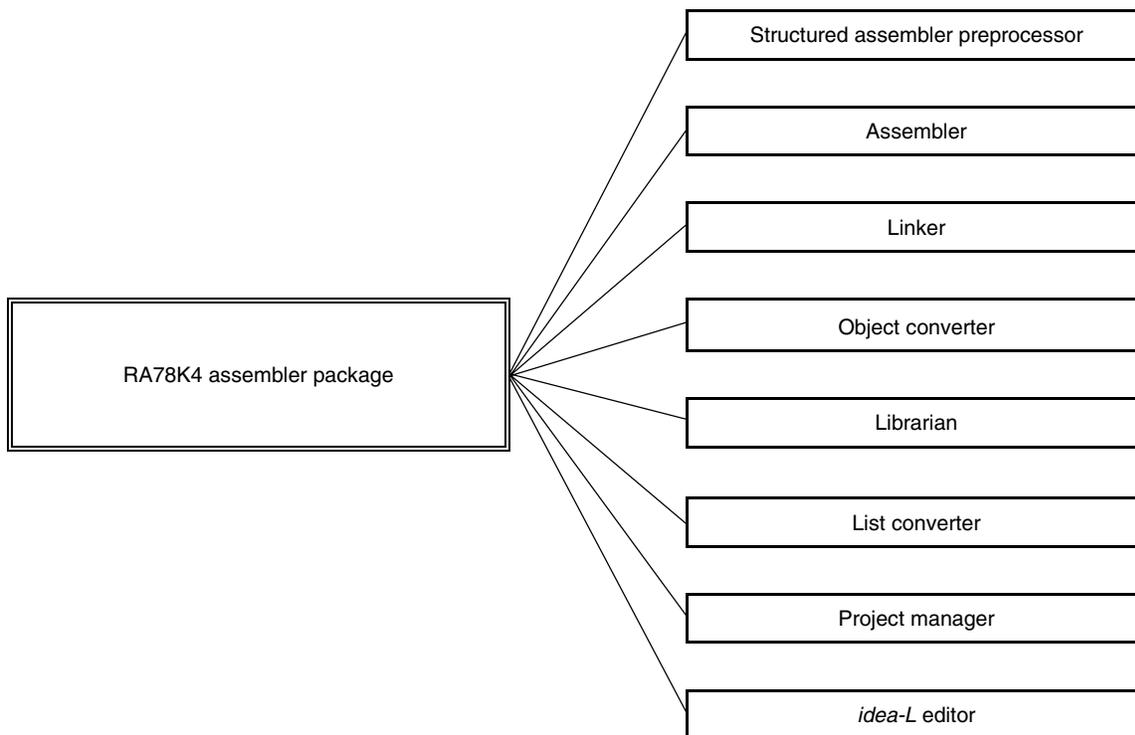
The RA78K4 Assembler Package is a generic term for a series of programs designed to translate source programs coded in the assembly language for 78K/IV Series microcontrollers into machine language coding.

The RA78K4 contains six programs: a structured assembler preprocessor, assembler, linker, object converter, librarian, and list converter.

In addition, a project manager that helps perform a series of operations including editing, compiling/assembling, linking, and debugging programs on Windows is also supplied with the RA78K4.

This project manager is supplied with an editor (*idea-L* editor).

Figure 1-1. RA78K4 Assembler Package



1.1.1 What is an assembler?

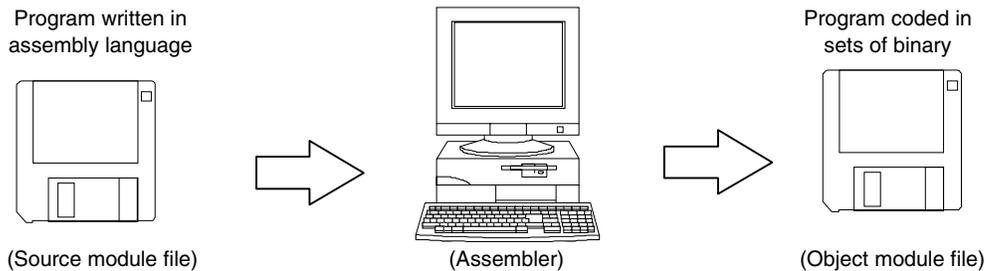
(1) Assembly language and machine language

An assembly language is the most fundamental programming language for a microcontroller.

For a microcontroller to do its job, programs and data are required. These programs and data must be written by people (i.e., programmers) and stored in the memory section of the microcontroller. The programs and data handled by the microcontroller are collections of binary numbers called machine language. For programmers, however, machine language code is difficult to remember, causing errors to occur frequently. Fortunately, methods exist whereby English abbreviations or mnemonics are used to represent the meanings of the original machine language codes in a way that is easy for people to comprehend. A programming language system that uses this symbolic coding is called an assembly language.

Since the microcontroller must handle programs in machine language form, another program is required that translates programs created in assembly language into machine language. This program is called an assembler.

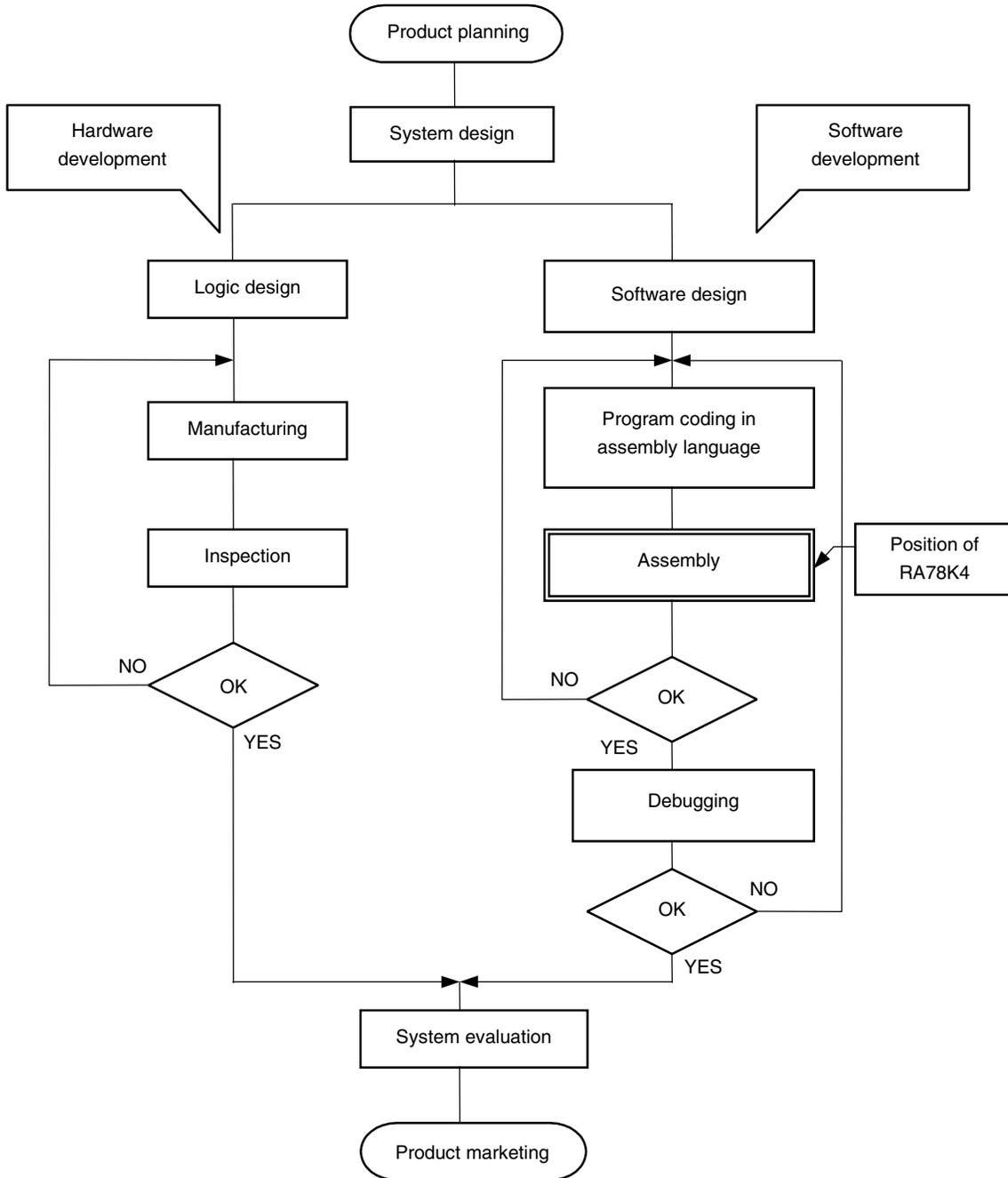
Figure 1-2. Flow of Assembler



(2) Development of products employing microcontrollers and role of RA78K4

Figure 1-3 Development Process of Products Employing Microcontrollers illustrates the position of assembly-language programming in the (software) product development process.

Figure 1-3. Development Process of Products Employing Microcontrollers



1.1.2 What is a relocatable assembler?

The machine language translated from a source language by the assembler is stored in the memory of the microcontroller before use. To do this, the location in memory where each machine language instruction is to be stored must already be determined. Therefore, information is added to the machine language assembled by the assembler, stating where in memory each machine language instruction is to be located.

Depending on the method of allocating addresses to machine language instructions, assemblers can be broadly divided into absolute assemblers and relocatable assemblers.

- **Absolute assembler**

An absolute assembler allocates machine language instructions assembled from the assembly language to absolute addresses.

- **Relocatable assembler**

In a relocatable assembler, the addresses determined for the machine language instructions assembled from the assembly language are tentative. Absolute addresses are determined subsequently by a program called the linker.

In the past, when a program was created with an absolute assembler, programmers generally had to complete programming at the same time. However, if all the components of a large program are created at the same time, the program becomes complicated, making analysis and maintenance of the program troublesome. To avoid this, such large programs are developed by dividing them into several subprograms, called modules, for each functional unit. This programming technique is called modular programming.

A relocatable assembler is an assembler suitable for modular programming.

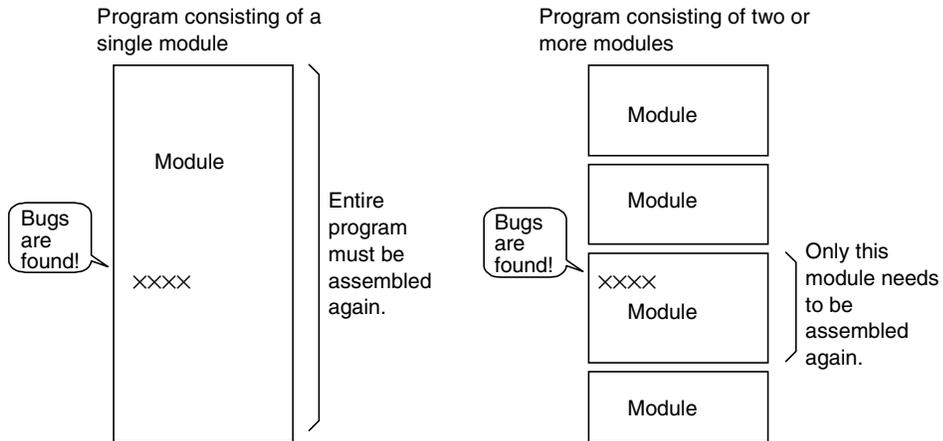
The following advantages can be derived from modular programming with a relocatable assembler:

(1) Increase in development efficiency

It is difficult to write a large program all at the same time. In such cases, dividing the program into modules for each function enables two or more programmers to develop subprograms in parallel to increase development efficiency.

Furthermore, if any bugs are found in the program, it is not necessary to assemble the entire program just to correct one part of the program; the bug can be corrected by assembling only the module that must be corrected. This shortens debugging time.

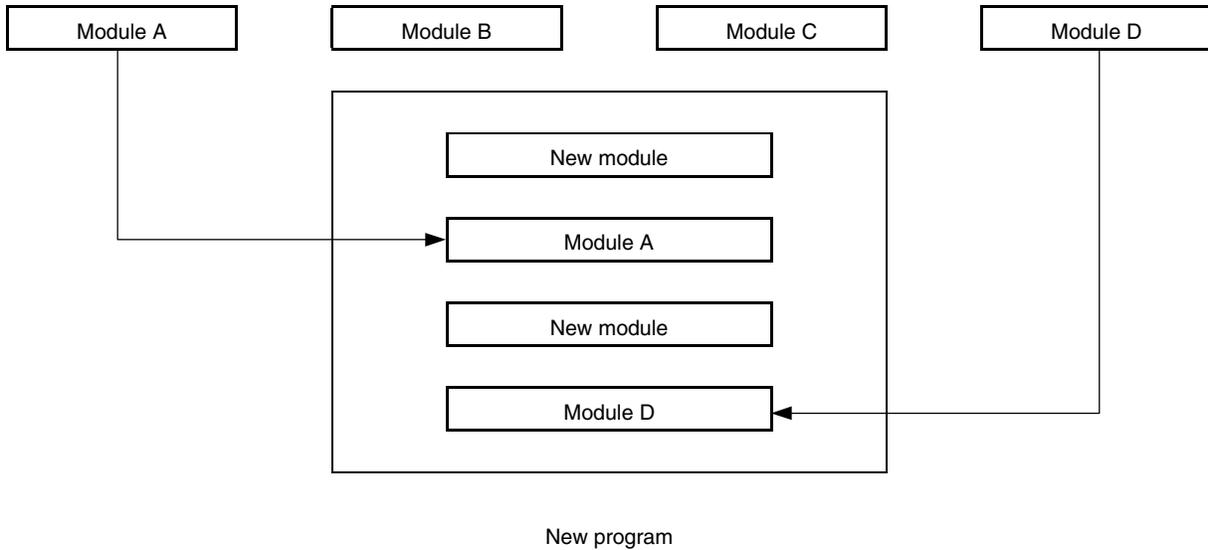
Figure 1-4. Reassembly for Debugging



(2) Utilization of resources

Highly reliable, highly versatile modules that have been previously created can be utilized for the creation of another program. By accumulating such highly versatile modules as software resources, considerable time and labor can be saved in developing a new program.

Figure 1-5. Program Development Using Existing Module



1.2 Reminders Before Program Development

Before beginning to develop a program, keep the following points in mind.

1.2.1 Maximum performance characteristics of RA78K4

(1) Maximum performance characteristics of assembler

Table 1-1. Maximum Performance Characteristics of Assembler

Item	Maximum Performance Characteristics	
	PC Version	WS Version
Number of symbols (local + public)	65,535 symbols ^{Note 1}	65,535 symbols ^{Note 1}
Number of symbols for which cross-reference list can be output	65,534 symbols ^{Note 2}	65,534 symbols ^{Note 2}
Maximum size of macro body for one macro reference	1 MB	1 MB
Total size of all macro bodies	10 MB	10 MB
Number of segments in one file	256 segments	256 segments
Macro and include specifications in one file	10,000	10,000
Macro and include specifications in one include file	10,000	10,000
Relocation data ^{Note 3}	65,535 items	65,535 items
Line number data	65,535 items	65,535 items
Number of BR directives in one file	32,767 directives	32,767 directives
Number of characters per line	2,048 characters ^{Note 4}	2,048 characters ^{Note 4}
Symbol length	256 characters	256 characters
Number of definitions of switch name ^{Note 5}	1,000	1,000
Character length of switch name ^{Note 5}	31 characters	31 characters
Number of nesting levels on include file in one file	8 levels	8 levels

- Notes**
1. XMS is used. If there is no XMS, a file is used.
 2. Memory is used. If there is no memory, a file is used.
 3. "Relocation data" is the data transferred to the linker when the assembler cannot decide the symbol values.
For example, when referring to an external reference symbol by a MOV instruction, two items of relocation data are generated in the .rel file.
 4. This includes the carriage return and feed codes. If 2,049 characters or more are described on a line, a warning message is output and the 2,049th and subsequent characters are ignored.
 5. The switch name is set to true or false by SET/RESET directives and used with \$IF, etc.

(2) Maximum performance characteristics of linker**Table 1-2. Maximum Performance Characteristics of Linker**

Item	Maximum Performance Characteristics	
	PC Version	WS Version
Number of symbols (local + public)	65,535 symbols	65,535 symbols
Line number data of same segment	65,535 items	65,535 items
Number of segments	65,535 segments	65,535 segments
Number of input modules	1,024 modules	1,024 modules

1.3 Features of RA78K4

The RA78K4 has the following features.

(1) Macro function

When the same group of instructions must be described in a source program over and over again, a macro can be defined by giving a single macro name to the group of instructions.

By using this macro function, the coding efficiency and readability of the program can be increased.

(2) Optimize function of branch instructions

The RA78K4 has a directive to automatically select a branch instruction (BR directive).

To create a program with high memory efficiency, a byte branch instruction must be described according to the branch destination range of the branch instruction. However, it is troublesome for the programmer to describe a branch instruction by paying attention to the branch destination range for each branching. By describing the BR directive, the assembler generates the appropriate branch instruction according to the branch destination range. This is called the optimize function of branch instructions.

(3) Conditional assembly function

With this function, a part of a source program can be specified for assembly or non-assembly according to a predetermined condition.

If a debug statement is described in a source program, whether or not the debug statement should be translated into machine language can be selected by setting a switch for conditional assembly. When the debug statement is no longer required, the source program can be assembled without major modifications to the program.

(4) General-purpose register name selection function

General-purpose registers can be described in terms of absolute names (R0, R1, RP0, etc.) and function names (X, A, AX, etc.). When describing function names in a source program, always describe the general-purpose register selection directive (RSS directive).

The RSS directive makes it possible to describe function names as general-purpose register identifiers in a source program.

CHAPTER 2 HOW TO DESCRIBE SOURCE PROGRAMS

This chapter describes the description methods, description formats, expressions and operators of the source program.

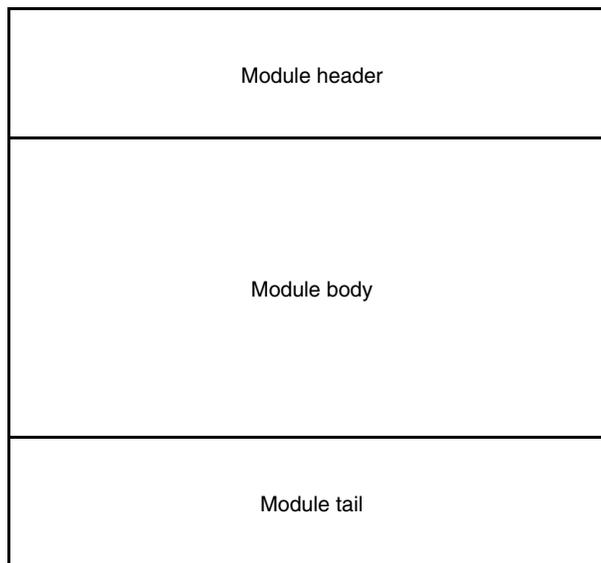
2.1 Basic Configuration of Source Program

When a source program is described by dividing it into several modules, each module that becomes the unit of input to the assembler is called a source module (if a source program consists of a single module, “source program” means the same as “source module”).

Each source module that becomes the unit of input to the assembler consists mainly of the following three parts.

- <1> Module header
- <2> Module body
- <3> Module tail

Figure 2-1. Configuration of Source Module



2.1.1 Module header

In the module header, the control instructions shown in **Table 2-1 Instructions That Can Be Described in Module Header** below can be described. Note that these control instructions can only be described in the module header.

Also, the module header can be omitted.

Table 2-1. Instructions That Can Be Described in Module Header

Item That Can Be Described	Explanation	Chapter/Section in This Manual
Control instructions that have the same functions as assembler options	Control instructions that have the same functions as assembler options are as follows: <ul style="list-style-type: none"> • PROCESSOR • XREF/NOXREF • DEBUG/NODEBUG/DEBUGA/NODEBUGA • TITLE • SYMLIST/NOSYMLIST • FORMFEED/NOFORMFEED • WIDTH • LENGTH • TAB • CHGSFR/CHGSFRA 	CHAPTER 4 CONTROL INSTRUCTIONS
Special control instructions output by high-level programs such as C compiler and structured assembler preprocessor	Special control instructions output by high-level programs such as C compiler and structured assembler preprocessor are as follows: <ul style="list-style-type: none"> • TOL_INF • DGS • DGL 	

2.1.2 Module body

In the module body, the following instructions cannot be described.

- Control instructions that have the same functions as assembler options

All other directives, control instructions, and instructions can be described in the module body.

The module body must be described by dividing it into units, called “segments”.

The user may define the following four segments with a directive corresponding to each segment.

- <1> Code segment.....Must be defined with the CSEG directive.
- <2> Data segment.....Must be defined with the DSEG directive.
- <3> Bit segmentMust be defined with the BSEG directive.
- <4> Absolute segment.....Must be defined by specifying a location address for the relocation attribute (AT location address) with the CSEG, DSEG, or BSEG directive. This segment may also be defined with the ORG directive.

The module body may be configured with any combination of segments.

However, a data segment and a bit segment should be defined before a code segment.

2.1.3 Module tail

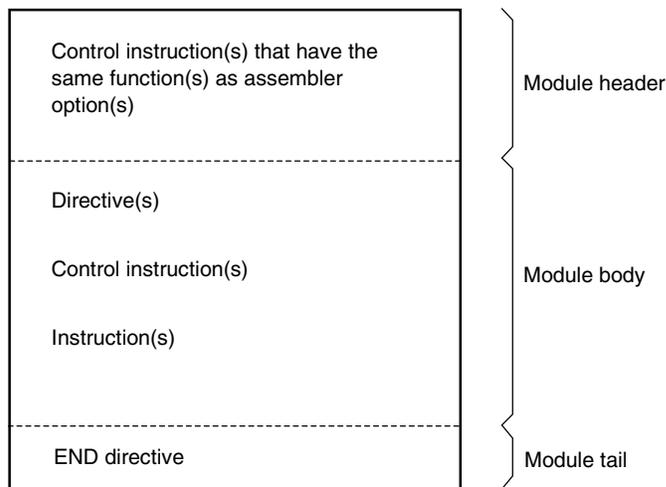
The module tail indicates the end of the source module. The END directive must be described in this part.

If anything other than a comment, a blank, a tab, or a line feed code is described following the END directive, the assembler will output a warning message and ignore the characters described after the END directive.

2.1.4 Overall configuration of source program

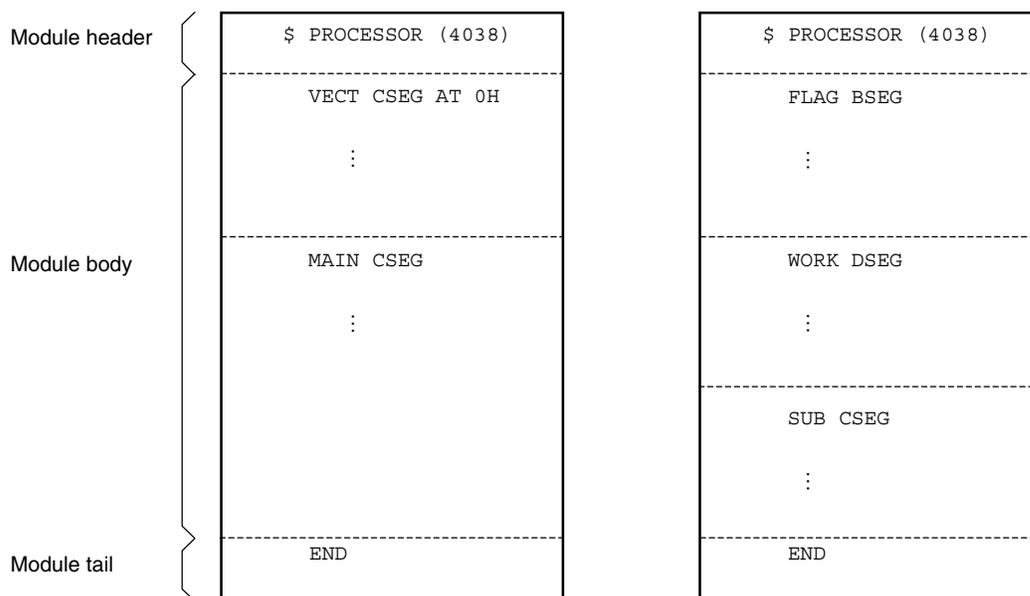
The overall configuration of a source module (source program) is as shown below.

Figure 2-2. Overall Configuration of Source Module



Examples of simple source module configurations are shown in Figure 2-3.

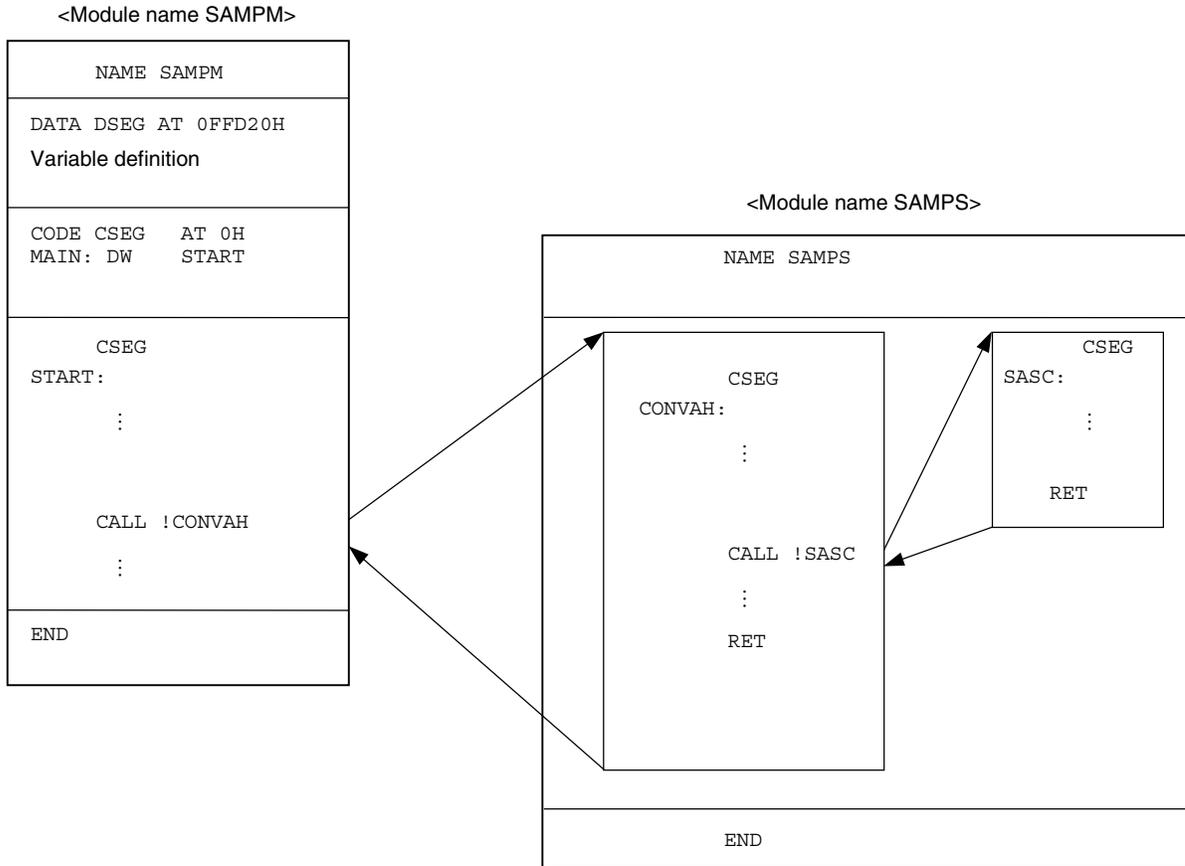
Figure 2-3. Examples of Source Module Configurations



2.1.5 Description example of source program

In this section, a description example of a source module (source program) is shown as a sample program. The configuration of the sample program can be illustrated simply as follows.

Figure 2-4. Configuration of Sample Program



This sample program was created by dividing a single source program into two modules. The module “SAMPM” is the main routine of this program and the module “SAMPS” is a subroutine to be called within the main routine.

<Main routine>

```

        NAME      SAMPM                                ; (1)
;*****
;*
;*      HEX -> ASCII Conversion Program              *
;*
;*          main-routine                            *
;*
;*****

        PUBLIC   MAIN,START                            ; (2)
        EXTRN   CONVAH                                ; (3)

DATA    DSEG     AT 0FFD20H                            ; (4)
HDTSA:  DS       1
STASC:  DS       2

CODE    CSEG     AT 0H                                ; (5)
MAIN:   DW       START

        CSEG                                          ; (6)
        LOCATION      15
START:  MOV       RFM,#00
        MOVG      SP,#0FFE00H
        MOV       MM,#00
        MOV       STBC,#08H

        MOV       HDTSA,#1AH
        MOVG      WHL,#HDTSA      ;set hex 2-code data in WHL register

        CALL     CONVAH      ;convert ASCII <- HEX
                                ;output BC-register <- ASCII code
        MOVG     TDE,#STASC    ;set DE <- store ASCII code table
        MOV      A,B
        MOV      [TDE+],A
        MOV      A,C
        MOV      [TDE+],A

        BR       $$

        END                                           ; (7)

```

- (1) Declaration of module name
- (2) Declaration of symbol referenced from another module as an external definition symbol
- (3) Definition of a symbol defined in another module as an external reference symbol
- (4) Declaration of the start of a data segment (to be located as an absolute segment starting from address 0FFD20H)
- (5) Declaration of the start of a code segment (to be located as an absolute segment starting from address 0H)
- (6) Declaration of the start of the code segment (meaning the end of the absolute segment)
- (7) Declaration of the end of the module

<Subroutine>

```

NAME      SAMPS                                     ; (8)
;*****
;*
;*  HEX -> ASCII Conversion Program                *
;*
;*          sub-routine                            *
;*
;*  input condition : (HL) <- hex 2 code          *
;*
;*  output condition ; BC-register <-ASCII 2 code *
;*
;*****

PUBLIC   CONVAH                                     ; (9)

CSEG                                          ; (10)
CONVAH:  MOV     A,#0
        ROL4    [WHL]          ;hex upper code load
        CALL   $!SASC
        MOV     B,A           ;store result

        MOV     A,#0
        ROL4    [WHL]          ;hex lower code load
        CALL   $!SASC
        MOV     C,A           ;store result

        RET

;*****
;* subroutine   convert ASCII code                *
;*  input      Acc (lower 4bits) <- hex code      *
;*  output     Acc          <- ASCII code         *
;*****

SASC:   CMP     A,#0AH         ;check hex code > 9
        BC     $SASC1
        ADD    A,#07H         ;bias(+7)
SASC1:  ADD     A,#30H         ;bias(+30)
        RET

END                                          ; (11)

```

- (8) Declaration of module name
- (9) Declaration of symbol referenced from another module as an external definition symbol
- (10) Declaration of the start of the code segment
- (11) Declaration of the end of the module

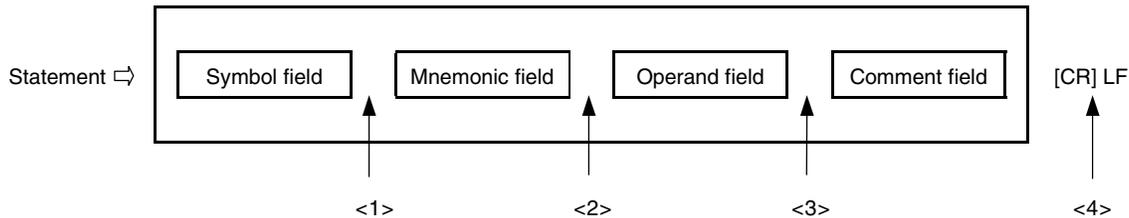
2.2 Description Format of Source Program

2.2.1 Configuration of statements

A source program consists of statements.

Each statement consists of the four fields shown in **Figure 2-5 Fields That Make Up a Statement**.

Figure 2-5. Fields That Make Up a Statement



- <1> The symbol field and the mnemonic field must be separated from each other with a colon (:) or one or more blanks or tabs (it depends on the instruction described in the mnemonic field whether colons or blanks are used).
- <2> The mnemonic field and the operand field must be separated from each other with one or more blanks or tabs. Depending on the instruction described in the mnemonic field, the operand field may not be required.
- <3> The comment field if used must be preceded with a semicolon (;).
- <4> Each line must be delimited with an LF code (one CR code may exist immediately before the LF code).

A statement must be described within a line. A maximum of 2,048 characters (including CR and LF) can be described per line.

Each TAB or independent CR is counted as a single character. If 2,049 or more characters are described, a warning message is output and the 2,049th and subsequent characters are ignored. However, 2,049 or more characters will be output to the assembly list.

An independent CR will not be output to the assembly list.

The following lines may also be described.

- Dummy line (line without statement description)
- Line consisting of the symbol field alone
- Line consisting of the comment field alone

2.2.2 Character set

Characters that can be described in a source file are classified into the following three types.

- Language characters
- Character data
- Comment characters

(1) Language characters

Language characters are characters used to describe instructions in a source program. The language character set includes alphabetic, numeric, and special characters.

[Alphanumeric Characters]

Name		Characters																				
Numeric characters		0	1	2	3	4	5	6	7	8	9											
Alphabetic characters	Uppercase letters	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	Lowercase letters	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
		v	w	x	y	z																

[Special Characters]

Character	Name	Main Use	
?	Question mark	Symbol equivalent to alphabetic characters	
@	Circa	Symbol equivalent to alphabetic characters	
_	Underscore	Symbol equivalent to alphabetic characters	
Blank HT (09H) , : ; CR (0DH) LF (0AH)	Tab code Comma Colon Semicolon Carriage return code Line-feed code	Delimiter symbols	Delimiter of each field Character equivalent to blank Delimiter of operands Delimiter of labels Symbol indicating the start of the Comment field Symbol indicating the end of a line (ignored in the assembler) Symbol indicating the end of a line
+ - * / . (,) < , > =	Plus sign Minus sign Asterisk Slash Period Left and right parentheses Not Equal sign Equal sign	Assembler operators	ADD operator or positive sign SUBTRACT operator or negative sign MULTIPLY operator • DIVIDE operator • Symbol indicating that operands with / are operated after reversing 0 and 1 to 1 and 0. Bit position specifier Symbols specifying the order of arithmetic operations to be performed Relational operators Relational operator
'	Single quotation mark	<ul style="list-style-type: none"> • Symbol indicating the start or end of a character constant • Symbol indicating a complete macro parameter 	
\$ \$! & # ! !! [%] []	Dollar sign Dollar sign and exclamation point Ampersand Sharp sign Exclamation point Two exclamation points Brackets and percent sign Brackets	<ul style="list-style-type: none"> • Symbol indicating the location counter • Symbol indicating the start of a control instruction equivalent to an assembler option • Symbol specifying relative addressing Symbol specifying relative (16-bit expression) addressing Concatenating symbol (used in macro body) Symbol specifying immediate addressing Symbol specifying absolute addressing Symbol indicating the start of absolute addressing in a 16-bit space range Symbol indicating an indirect 3-byte operation Symbol specifying indirect addressing	

(2) Character data

“Character data” refers to characters used to describe string constants, character strings, and control instructions (TITLE, SUBTITLE, INCLUDE).

[Character set for character data]

- All characters except “00H” can be used, codes may be different depending on the operating system. If “00H” has been described, an error will result and subsequent characters before the closing single quotation mark (') will be ignored.
- If any illegal character has been described, the assembler will replace the illegal character with “!” for output to the assembly list (an independent CR (0DH) code will not be output to the assembly list).
- With Windows, the assembler interprets the code “1AH” as the end of the file (EOF) and thus the code cannot be a part of the input data.

(3) Comment characters

“Comment characters” refers to characters used to describe a comment statement.

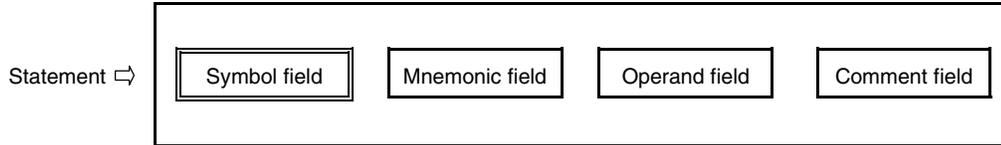
[Character set for comments]

- Characters that can be used in a comment statement are the same as those in the character set for character data. However, no error will result even if the code “00H” has been described. Instead, the assembler will output the illegal character to the assembly list replacing it with “!”.

2.2.3 Fields that make up a statement

This subsection details the respective fields that make up a statement.

(1) Symbol field



A symbol is described in the symbol field. The term “symbol” refers to a name given to numerical data or an address.

By using symbols, the contents of a source program can be understood more easily.

[Symbol types]

Symbols are classified into the types shown in **Table 2-2**, depending on their use and method of definition.

Table 2-2. Symbol Types

Symbol Type	Use	Method of Definition
Name	Used as numerical data or an address in a source program.	This type is described in the symbol field of the EQU, SET, or DBIT directive.
Label	Used as address data in a source program.	This type is defined by suffixing a colon (:) to a symbol.
External reference name	Used to reference a symbol defined by a module by another module.	This type is described in the operand field of the EXTRN or EXTBIT directive.
Segment name	Symbol used during linker operation	This type is defined in the symbol field of the CSEG, DSEG, BSEG or ORG directive.
Module name	Used during symbolic debugging	This type is described in the operand field of the NAME directive.
Macro name	Used for macro reference in a source program.	This type is described in the symbol field of the MACRO directive.

[Conventions of symbol description]

All symbols must be described according to the following rules.

- <1> A symbol must be made up of alphanumeric characters and special characters (? , @ , and _) that can be used as characters equivalent to alphabetic characters.
None of the numerals 0 to 9 can be used as the first character of a symbol.
- <2> A symbol must be made up of not more than 31 characters. Characters in excess of the maximum symbol length will be ignored.
- <3> No reserved word can be used as a symbol. Reserved words are indicated in **APPENDIX A LIST OF RESERVED WORDS**.
- <4> The same symbol cannot be defined more than once (however, a name defined with the SET directive can be redefined with the SET directive).
- <5> The assembler distinguishes between lowercase and uppercase characters.
- <6> When describing a label in the Symbol field, ":" (colon) must be described immediately after the label.

(Examples of correct symbol descriptions)

```

CODE01      CSEG      ; "CODE01" is a segment name.
VAR01      EQU  10H    ; "VAR01" is a name.
LAB01:     DW   0      ; "LAB01" is a label.
           NAME SAMPLE ; "SAMPLE" is a module name.
MAC1      MACRO      ; "MAC1" is a macro name.
    
```

(Examples of incorrect symbol descriptions)

```

1ABC      EQU  3      ; A numeral cannot be used as the 1st character of a
                    ; symbol.
LAB      MOV  A, R0    ; "LAB" is a label and must be separated from the Mnemonic
                    ; field with a colon (:).
FLAG:     EQU  10H    ; A colon (:) is not necessary in a name.
    
```

(Example of a symbol that is too long)

```

A123456789B12 ~ Y1234567890123456      EQU  70H
    250                                ; Character "6", which is in excess of the maximum symbol length (256
    ; characters) is ignored. The symbol will be defined as
    ; "A123456789B12 ~ Y123456789012345".
    
```

(Example of a statement composed of a symbol only)

```

ABCD:     ; "ABCD" will be defined as a label.
    
```

[Cautions about symbols]

The symbol "??RAnnn (n = 0000 to FFFF)" is a symbol that is automatically replaced by the assembler every time a local symbol is expanded inside a macro body. Be careful not to define this symbol twice.

When a segment name is not specified by a segment definition directive, the assembler generates a segment name automatically. These segments are shown in **Table 2-3 Names of Segments Automatically Generated by Assembler**.

Duplicate segment name definition causes an error.

Table 2-3. Names of Segments Automatically Generated by Assembler

Segment name	Directive	Relocation Attribute	
?An	ORG directive	n = 000000 to FFFFFFFF	
?CSEG	CSEG directive	UNIT	
?CSEGT0		CALLT0	
?CSEGFx		FIXED	
?CSEGFxA		FIXEDA	
?CSEGB		BASE	
?CSEGP		PAGE	
?CSEGP64		PAGE64K	
?DSEG		DSEG directive	UNIT
?DSEGS	SADDR		
?DSEGSP	SADDRP		
?DSEGS2	SADDR2		
?DSEGSP2	SADDRP2		
?DSEGSA	SADDRA		
?DSEGDT	DTABLE		
?DSEGDTP	DTABLEP		
?DSEGP	PAGE		
?DSEGP64	PAGE64K		
?DSEGG	GRAM		
?BSEG	BSEG directive		UNIT
?BSEGUP			UNITP
?BSEGS		SADDR	
?BSEGSP		SADDRP	
?BSEGS2		SADDR2	
?BSEGSP2		SADDRP2	
?BSEGSA		SADDRA	
?BSEGG		GRAM	

[Symbol attributes]

All names and labels have both a value and an attribute.

The value refers to the value of defined numerical data or address data itself.

Segment names, module names, and macro names do not have a value.

The attribute of a symbol is called a symbol attribute and must be one of the eight types indicated in **Table 2-4 Symbol Attributes and Values**.

Table 2-4. Symbol Attributes and Values

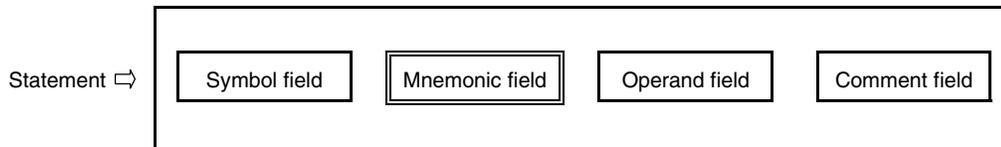
Attribute Type	Classification	Value
NUMBER	<ul style="list-style-type: none"> Names to which numeric constants are assigned Symbols defined with the EXTRN directive Numeric constants 	Decimal representation: 0 to 65535 Hexadecimal representation: 0H to FFFFH
ADDRESS	<ul style="list-style-type: none"> Symbols defined as labels Names defined as labels with EQU and SET directives 	
BIT	<ul style="list-style-type: none"> Names defined as bit values Names within BSEG Symbols defined with the EXTBIT directive 	saddr area
CSEG	Segment names defined with the CSEG directive	These attribute types have no value.
DSEG	Segment names defined with the DSEG directive	
BSEG	Segment names defined with the BSEG directive	
MODULE	Module names defined with the NAME directive (a module name if not defined is created from the primary name of the input source filename)	
MACRO	Macro names defined with the MACRO directive	

Examples

```

TEN      EQU      10H      ; Name "TEN" has attribute "NUMBER" and value "10H".
                ORG      80H
START:   MOV      A, #10H  ; Label "START" has attribute "ADDRESS" and value "80H".
BIT1    EQU      0FE20H.0 ; Name "BIT1" has attribute "BIT" and value "0FE20H.0".

```

(2) Mnemonic field

In the mnemonic field, a mnemonic instruction, a directive, or a macro reference is described.

For an instruction or directive requiring an operand or operands, the mnemonic field must be separated from the operand field with one or more blanks or tabs.

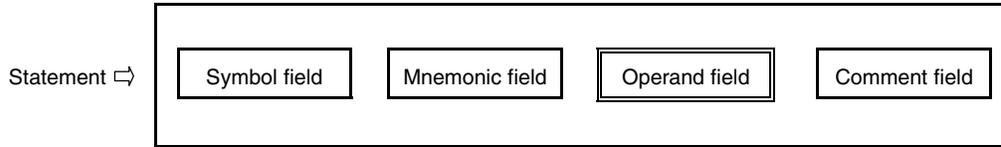
However, for the first operand of an instruction that begins with “#”, “\$”, “!”, “[”, “[%”, “&”, “!!”, or “\$!”, assembly will be executed properly even if nothing exists between the mnemonic field and the first operand field.

(Examples of correct descriptions)

```
MOV      A, #0H
CALL     !CONVAH
RET
```

(Examples of incorrect descriptions)

```
MOVA     #0H      ; No blank exists between the mnemonic and operand fields.
C  ALL   ! CONVAH ; A blank exists within the mnemonic field.
ZZZ      ; The 78K/IV Series has no such instruction as “ZZZ”.
```

(3) Operand field

In the operand field, the data (operands) required for executing the instruction, directive, or macro reference is described.

Depending on the instruction or directive, no operand is required in the operand field or two or more operands must be described in the operand field.

When describing two or more operands, delimit each operand with a comma (,).

The following types of data can be described in the operand field.

- Constants
- Character strings
- Register names
- Special characters
- Relocation attribute names of segment definition directives
- Symbols
- Expressions
- Bit terms
- Macro service control word

The size and attribute of the required operand may differ depending on the instruction or directive. Refer to **2.5 Characteristics of Operands** for the sizes and attributes of operands.

For the operand representation formats and description methods in the instruction set, see the user's manual of the microcontroller for which software is being developed.

Each of the data types that can be described in the operand field is detailed below.

[Constants]

A constant is a fixed value or data item and is also referred to as immediate data.

Constants are divided into numeric constants and character-string constants.

- Numeric constants

A binary, octal, decimal, or hexadecimal number can be described as a numeric constant. The method of representing each numeric constant type is shown in **Table 2-5** below.

A numeric constant will be processed as unsigned 24-bit data.

Value range: $0 \leq n \leq 16,777,215$ (0FFFFFFH)

When describing a negative value, use the minus sign of the operator.

Table 2-5. Methods of Representing Numeric Constants

Constant	Method of Representation	Example
Binary constant	<ul style="list-style-type: none"> Character "B" or "Y" is suffixed to a numerical value. 	1101B 1101Y
Octal constant	<ul style="list-style-type: none"> Character "O" or "Q" is suffixed to a numerical value. 	74O 74Q
Decimal constant	<ul style="list-style-type: none"> A numerical value is described as is, or character "D" or "T" is suffixed to a numerical value. 	128 128D 128T
Hexadecimal constant	<ul style="list-style-type: none"> Character "H" is suffixed to a numerical value. If the first character begins with "A", "B", "C", "D", "E", or "F", "0" must be prefixed to the constant. 	8CH 0A6H

- Character-string constants

A character-string constant is expressed by enclosing a string of characters from those shown in **2.2.2 Character set** in a pair of single quotation marks (').

As a result of an assembly process, the character-string constant is converted into 7-bit ASCII code with the parity bit (MSB) set as "0".

The length of a string constant is 0 to 3 characters.

To use the single quotation mark itself as a string constant, the single quotation mark must be input twice in succession.

Examples of character-string constant descriptions:

```
'A'           ; Represents "41H"
' '           ; Represents "20H"
' ' ' '       ; Represents "27H"
' ' 'A'       ; Represents "2741H"
' 'AA'       ; Represents "274141H"
```

[Character strings]

A character string is expressed by enclosing a string of characters from those shown in **2.2.2 Character set** in a pair of single quotation marks ('). Character strings are mainly used for operands in the DB directive and TITLE or SUBTITLE control instruction.

- Application examples of character strings

```
                CSEG
MAS1:  DB      'YES'    ; Initializes with character string "YES".
MAS2:  DB      'NO'    ; Initializes with character string "NO".
```

[Register names]

The following registers can be described in the operand field.

- General-purpose registers
- General-purpose register pairs
- 3-byte registers
- Special function registers

General-purpose registers and general-purpose register pairs can be described with their absolute names (R0 to R15 and RP0 to RP7), as well as with their function names (X, A, B, C, D, E, H, L, AX, BC, DE, HL, VP, UP).

The register names that can be described in the operand field may be different depending on the type of instruction. For details of the method of describing each register name, see the user's manual of the microcontroller for which software is being developed.

[Special characters]

Special characters that can be described in the operand field are shown in **Table 2-6 Special Characters That Can Be Described in Operand Field**.

Table 2-6. Special Characters That Can Be Described in Operand Field

Special Character	Function
\$	<ul style="list-style-type: none"> Indicates the location address of the instruction having this operand (or the 1st byte of this address, in the case of addresses with a multiple-byte instruction). Indicates a relative (8-bit) addressing mode for a Branch instruction or a Call instruction.
!\$	<ul style="list-style-type: none"> Indicates a relative (16-bit) addressing mode for a Branch instruction or a Call instruction.
!	<ul style="list-style-type: none"> Indicates an absolute (16-bit) addressing mode for a Branch instruction or a Call instruction. Indicates the specification of addr16 which allows all memory space to be specified with an MOV instruction.
!!	<ul style="list-style-type: none"> Indicates a 24/20 absolute addressing mode for a Branch instruction or a Call instruction. Indicates the specification of addr24/addr20 which allows all memory space to be specified with an MOV instruction.
#	<ul style="list-style-type: none"> Indicates immediate data.
[]	<ul style="list-style-type: none"> Indicates indirect addressing mode.
[%]	<ul style="list-style-type: none"> Indicates indirect addressing mode and 3-byte instruction.

- Application examples of special characters

Address	Source program
100	ADD R15, R1
101	LOOP: INC R1
103	BR \$\$-2 ...<1>
106	BR !\$+100H ...<2>

<1> The second \$ in the operand indicates address 103H. Describing “BR \$LOOP” results in the same operation.

<2> The second \$ in the operand indicates address 106H.

[Relocation attributes of segment definition directives]

Relocation attributes can be described in the operand field.

For details of relocation attributes, refer to **3.2 Segment Definition Directives**.

[Symbols]

If a symbol is described in the operand field, an address (or value) allocated to that symbol becomes the operand value.

- Application examples of symbols

VALUE	EQU	12345678H	
	MOVD	RP0, VALUE	; This description means the same as “MOV D RP0, 12345678H”.

[Expressions]

An expression is a constant, \$ (which indicates a location address), or symbol connected with an operator.

An expression can be described where numeric values can be expressed as instruction operands.

For details of expressions and operators, refer to **2.3 Expressions and Operators**.

- Examples of expressions

```
TEN    EQU    10H
      MOV    A, #TEN-5H
```

In this example, "TEN-5H" is an expression.

In this expression, the name and numeric constant are connected with a – (minus) operator. The value of the expression is BH.

Therefore, this description can be rewritten as "MOV A, #0BH".

[Bit terms]

A bit term can be obtained by the bit position specifier. For details of bit terms, refer to **2.4 Bit Position Specifier**.

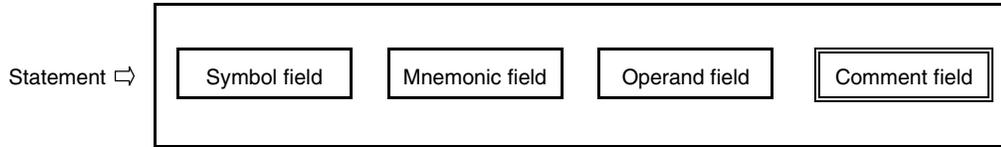
- Examples of bit terms

```
CLR1   A.5
SET1   1+0FE30H.3 ; The operand value is 0FE31H.3.
CLR1   0FE40H.4+2 ; The operand value is 0FE40H.6.
```

[Macro Service Control Word]

Refer to the user's manual of each device for the macro service control word which can be described in an operand.

Caution The macro service control word is processed as an absolute value. Therefore, when the SFR area change option (-CSA) or the SFR area change control instruction (\$CHGSFRA) is specified, it may not be possible to access the macro service control word using the direct addressing instruction specified in the operand.

(4) Comment field

In the comment field, comments or remarks may be described following the input of a semicolon (;). The comment field is from a semicolon to the line-feed code of that line or EOF. By describing a comment statement in the comment field, an easy-to-understand source program can be created. The comment statement in the comment field is not subject to assembler operation (i.e., conversion into machine language) but will be output without change on an assembly list.

Characters that can be described in the comment field are those shown in **2.2.2 Character set**.

(Examples of comments)

```

NAME      SAMPM
;*****
;*
;*      HEX -> ASCII Conversion Program      *
;*
;*      main-routine                          *
;*
;*****

PUBLIC   MAIN, START
EXTRN   CONVAH

DATA    DSEG    AT 0FFD20H
HDTSA:  DS      1
STASC:  DS      2

CODE    CSEG    AT 0H
MAIN:   DW      START

CSEG
LOCATION 15
START:  MOV      RFM, #00
        MOVG     SP, #0FFE00H
        MOV      MM, #00
        MOV      STBC, #08H
        MOV      HDTSA, #1AH
        MOVG     WHL, #HDTSA      ;set hex 2-code data in WHL
                                   register
        CALL     CONVAH          ;convert ASCII <- HEX
                                   ;output BC-register <- ASCII
                                   code
        MOVG     TDE, #STASC     ;set DE <- store ASCII code
                                   table

        MOV      A, B
        MOV      [TDE+], A
        MOV      A, C
        MOV      [TDE+], A
        BR      $$
        END

```

↑
Lines consisting of
comment field only
↓

↑
Lines in which
comments are
described in
comment field
↓

2.3 Expressions and Operators

An expression is a symbol, constant, location address (indicated by \$) or bit term with an operator attached, or combined by one or more operators.

Elements of an expression other than the operators are called terms, and are referred to as the 1st term, 2nd term, and so forth from left to right, in the order of their description.

Operators are available in the types shown in **Table 2-7 Types of Operators**, and the order of their precedence in calculation has been predetermined as shown in **Table 2-8 Order of Precedence of Operators**.

Parentheses “()” are used to change the order in which calculations are performed.

Example: MOV A, #5* (SYM+1) ; <1>

In <1> above, “5* (SYM+1)” is an expression. “5” is the 1st term of the expression and “SYM” and “1” are the 2nd and 3rd terms respectively. “*”, “+”, and “()” are operators.

Table 2-7. Types of Operators

Type of Operator	Operators
Arithmetic operators	+ sign, - sign, +, -, *, /, MOD
Logical operators	NOT, AND, OR, XOR
Relational operators	EQ or =, NE or < >, GT or >, GE or >=, LT or <, LE or <=
Shift operators	SHR, SHL
Byte-separating operators	HIGH, LOW
Word-separating operators	HIGHW, LOWW
Special operators	DATAPOS, BITPOS, MASK
Other operators	()

The above operators can also be divided into unary operators, special unary operators, binary operators, N-ary operators, and other operators.

Unary operators: + sign, - sign, NOT, HIGH, LOW, HIGHW, LOWW

Special unary operators: DATAPOS, BITPOS

Binary operators: +, -, *, /, MOD, AND, OR, XOR, EQ or =, NE or < >, GT or >, GE or >=, LT or <, LE or <=, SHR, SHL

N-ary operators: MASK

Other operators: ()

Table 2-8. Order of Precedence of Operators

Priority	Priority Level	Operators
Higher ↑ ↓ Lower	1	+ sign, – sign, NOT, HIGH, LOW, DATAPOS, BITPOS, MASK
	2	*, /, MOD, SHR, SHL
	3	+, –
	4	AND
	5	OR, XOR
	6	EQ or =, NE or <>, GT or >, GE or >=, LT or <, LE or <=

Operations on expressions are performed according to the following rules.

- <1> Operations are performed according to the order of precedence given to each operator. If two or more operators of the same order of precedence exist in an expression, the operation designated by the leftmost operator will be carried out. In the case of unary operators, the operation will be performed from right to left.
- <2> An expression in parentheses is carried out before expressions outside the parentheses.
- <3> Operations between two or more unary operators are allowed.

Examples: 1==–1==1
 –1==+1==–1

- <4> Expressions are calculated within 32 bits, without signs. If an overflow occurs in operation due to an expression exceeding 32 bits, the overflowed value is ignored.
- <5> If a constant exceeds 24 bits (0FFFFFFH), an error will result and the value of the result will be regarded as 0 for calculation.
- <6> In division, the decimal fraction part of the result will be truncated. If the divisor is 0, an error will occur, and the result will be 0.

2.3.1 Functions of operators

The functions of the respective operators are described in this section.

(1) + (ADD) operator**[Function]**

Returns the sum of the values of the 1st and 2nd terms of an expression.

[Application example]

```

                                ORG    100H
START:                          BR     !$+6      ; (a)
                                :
```

[Explanation]

The BR instruction causes a jump to “current location address plus 6”, namely, to address “100H+6H=106H”.

Therefore, (a) in the above example can also be described as: START: BR !106H

(2) – (SUBTRACT) operator**[Function]**

Returns the result of subtraction of the 2nd-term value from the 1st-term value.

[Application example]

```

                                ORG    100H
BACK:                            BR     BACK-6H   ; (b)
                                :
```

[Explanation]

The BR instruction causes a jump to “address assigned to BACK minus 6”, namely, to address “100H-6H=0FAH”.

Therefore, (b) in the above example can also be described as: BACK: BR !0FAH

(3) * (MULTIPLY) operator**[Function]**

Returns the result of multiplication (product) between the values of the 1st and 2nd terms of an expression.

[Application example]

```

TEN      EQU      10H
          MOV      A, #TEN*3      ; (c)
          :
```

[Explanation]

With the EQU directive, the value "10H" is defined in the name "TEN".

"#" indicates immediate data. The expression "TEN*3" is the same as "10H*3" and returns the value "30H".

Therefore, (c) in the above expression can also be described as: MOV A,#30H

(4) / (DIVIDE) operator**[Function]**

Divides the value of the 1st term of an expression by the value of its 2nd term and returns the integer part of the result. The decimal fraction part of the result will be truncated. If the divisor (2nd term) of a division operation is 0, an error will result.

[Application example]

```

          MOV      A, #256/50      ; (d)
```

[Explanation]

The result of the division "256/50" is 5 with remainder 6.

The operator returns the value "5" that is the integer part of the result of the division.

Therefore, (d) in the above expression can also be described as: MOV A,#5

(5) MOD (Remainder) operator**[Function]**

Obtains the remainder in the result of dividing the value of the 1st term of an expression by the value of its 2nd term.

An error will result if the divisor (2nd term) is 0.

A blank is required before and after the MOD operator.

[Application example]

```
MOV    A, #256 MOD 50 ; (e)
```

[Explanation]

The result of the division "256/50" is 5 with remainder 6.

The MOD operator returns the remainder 6.

Therefore, (e) in the above expression can also be described as: MOV A,#6.

(6) + sign**[Function]**

Returns the value of the term of an expression without change.

[Application example]

```
FIVE   EQU   +5
```

[Explanation]

The value "5" of the term is returned without change.

The value "5" is defined in name "FIVE" with the EQU directive.

(7) – sign**[Function]**

Returns the value of the term of an expression by the two's complement.

[Application example]

```
NO     EQU   -1
```

[Explanation]

-1 becomes the two's complement of 1.

The two's complement of binary 0000 0000 0000 0001

becomes: 1111 1111 1111 1111

Therefore, with the EQU directive, the value "0FFFFH" is defined in the name "NO".

(1) NOT operator (negation)**[Function]**

Negates the value of the term of an expression on a bit-by-bit basis and returns the result.

A blank is required between the NOT operator and the term.

[Application example]

```
MOVW AX, #NOT 3H ; (a)
```

[Explanation]

Logical negation is performed on "3H" as follows:

```
NOT) 0000 0000 0000 0000 0000 0011
```

```
1111 1111 1111 1111 1111 1100
```

0FFFCH is returned.

Therefore, (a) can also be described as: MOVW AX, #0FFFCH

(2) AND operator (logical product)**[Function]**

Performs an AND (logical product) operation between the value of the 1st term of an expression and the value of its 2nd term on a bit-by-bit basis and returns the result.

A blank is required before and after the AND operator.

[Application example]

```
MOV A, #6FAH AND 0FH ; (b)
```

[Explanation]

AND operation is performed between the two values "6FAH" and "0FH" as follows:

```
0000 0000 0000 0110 1111 1010
```

```
AND) 0000 0000 0000 0000 0000 1111
```

```
0000 0000 0000 1010 0000 1010
```

The result 0AH is returned. Therefore, (b) in the above expression can also be described as: MOV A, #0AH

(3) OR operator (logical sum)**[Function]**

Performs an OR (logical sum) operation between the value of the 1st term of an expression and the value of its 2nd term on a bit-by-bit basis and returns the result.

A blank is required before and after the OR operator.

[Application example]

```
MOV   A, #0AH OR  1101B  ; (c)
```

[Explanation]

OR operation is performed between the two values "0AH" and "1101B" as follows:

```
      0000 0000 0000 0000 0000 1010
OR)   0000 0000 0000 0000 0000 1101
-----
      0000 0000 0000 0000 0000 1111
```

The result 0FH is returned. Therefore, (c) in the above expression can also be described as: MOV A, #0FH

(4) XOR operator (exclusive logical sum)**[Function]**

Performs an exclusive-OR operation between the value of the 1st term of an expression and the value of its 2nd term on a bit-by-bit basis and returns the result.

A blank is required before and after the XOR operator.

[Application example]

```
MOV   A, #9AH XOR  9DH  ; (d)
```

[Explanation]

XOR operation is performed between the two values "9AH" and "9DH" as follows:

```
      0000 0000 0000 0000 1001 1010
XOR)  0000 0000 0000 0000 1001 1101
-----
      0000 0000 0000 0000 0000 0111
```

The result 7H is returned. Therefore, (d) in the above expression can also be described as: MOV A, #7H

(1) EQ or = (equal) operator**[Function]**

Returns 0FFH (true) if the value of the 1st term of an expression is equal to the value of its 2nd term, and 00H (false) if both values are not equal.

A blank is required before and after the EQ operator.

[Application example]

A1	EQU	12C4H			
A2	EQU	12C0H			
	MOV	A, #A1	EQ	(A2+4H)	; (a)
	MOV	X, #A1	EQ	A2	; (b)

[Explanation]

In (a) above, the expression "A1 EQ (A2+4H)" becomes "12C4H EQ (12C0H+4H)".

The operator returns 0FFH because the value of the 1st term is equal to the value of the 2nd term.

In (b) above, the expression "A1 EQ A2" becomes "12C4H EQ 12C0H".

The operator returns 00H because the value of the 1st term is not equal to the value of the 2nd term.

(2) NE or < > (not equal) operator**[Function]**

Returns 0FFH (true) if the value of the 1st term of an expression is not equal to the value of its 2nd term, and 00H (false) if both values are equal.

A blank is required before and after the NE operator.

[Application example]

A1	EQU	5678H			
A2	EQU	5670H			
	MOV	A, #A1	NE	A2	; (c)
	MOV	A, #A1	NE	(A2+8H)	; (d)

[Explanation]

In (c) above, the expression "A1 NE A2" becomes "5678H NE 5670H".

The operator returns 0FFH because the value of the 1st term is not equal to the value of the 2nd term.

In (d) above, the expression "A1 NE (A2+8H)" becomes "5678H NE (5670H+8H)".

The operator returns 00H because the value of the 1st term is equal to the value of the 2nd term.

(3) GT or > (greater than) operator**[Function]**

Returns 0FFH (true) if the value of the 1st term of an expression is greater than the value of its 2nd term, and 00H (false) if the value of the 1st term is equal to or less than the value of the 2nd term.

A blank is required before and after the GT operator.

[Application example]

A1	EQU	1023H			
A2	EQU	1013H			
	MOV	A, #A1	GT	A2	; (e)
	MOV	X, #A1	GT	(A2+10H)	; (f)

[Explanation]

In (e) above, the expression "A1 GT A2" becomes "1023H GT 1013H".

The operator returns 0FFH because the value of the 1st term is greater than the value of the 2nd term.

In (f) above, the expression "A1 GT (A2+10H)" becomes "1023H GT (1013H+10H)".

The operator returns 00H because the value of the 1st term is equal to the value of the 2nd term.

(4) GE or >= (greater-than or equal) operator**[Function]**

Returns 0FFH (true) if the value of the 1st term of an expression is greater than or equal to the value of its 2nd term, and 00H (false) if the value of the 1st term is less than the value of the 2nd term.

A blank is required before and after the GE operator.

[Application example]

A1	EQU	2037H			
A2	EQU	2015H			
	MOV	A, #A1	GE	A2	; (g)
	MOV	X, #A1	GE	(A2+23H)	; (h)

[Explanation]

In (g) above, the expression "A1 GE A2" becomes "2037H GE 2015H".

The operator returns 0FFH because the value of the 1st term is greater than the value of the 2nd term.

In (h) above, the expression "A1 GE (A2+23H)" becomes "2037H GE (2015H+23H)".

The operator returns 00H because the value of the 1st term is less than the value of the 2nd term.

(5) LT or < (less than) operator**[Function]**

Returns 0FFH (true) if the value of the 1st term of an expression is less than the value of its 2nd term, and 00H (false) if the value of the 1st term is equal to or greater than the value of the 2nd term.

A blank is required before and after the LT operator.

[Application example]

A1	EQU	1000H		
A2	EQU	1020H		
	MOV	A, #A1	LT A2	; (i)
	MOV	X, # (A1+20H)	LT A2	; (j)

[Explanation]

In (i) above, the expression "A1 LT A2" becomes "1000H LT 1020H".

The operator returns 0FFH because the value of the 1st term is less than the value of the 2nd term.

In (j) above, the expression "(A1+20H) LT A2" becomes "(1000H+20H) LT 1020H".

The operator returns 00H because the value of the 1st term is equal to the value of the 2nd term.

(6) LE or <= (less than or equal) operator**[Function]**

Returns 0FFH (true) if the value of the 1st term of an expression is less than or equal to the value of its 2nd term, and 00H (false) if the value of the 1st term is greater than the value of the 2nd term.

A blank is required before and after the LE operator.

[Application example]

A1	EQU	103AH		
A2	EQU	1040H		
	MOV	A, #A1	LE A2	; (k)
	MOV	X, # (A1+7H)	LE A2	; (l)

[Explanation]

In (k) above, the expression "A1 LE A2" becomes "103AH LE 1040H".

The operator returns 0FFH because the value of the 1st term is less than the value of the 2nd term.

In (l) above, the expression "(A1+7H) LE A2" becomes "(103AH+7H) LE 1040H".

The operator returns 00H because the value of the 1st term is greater than the value of the 2nd term.

(1) SHR (shift right) operator**[Function]**

Returns a value obtained by shifting the value of the 1st term of an expression to the right the number of bits specified by the value of the 2nd term. Zeros equivalent to the specified number of bits shifted move into the higher bits.

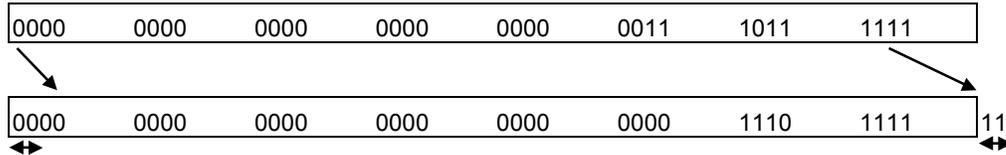
A blank is required before and after the SHR operator.

[Application example]

```
MOVW  RPL, #0003BFH  SHR  2    ; (a)
```

[Explanation]

This operator shifts the value "0003BFH" to the right by 2 bits.



0's are inserted.

Right-shifted by 2 bits.

The value "0000EFH" is returned.

Therefore, (a) in the above example can also be described as: MOV A, #0EFH

(2) SHL (shift left) operator**[Function]**

Returns a value obtained by shifting the value of the 1st term of an expression to the left the number of bits specified by the value of the 2nd term. Zeros equivalent to the specified number of bits shifted move into the higher bits.

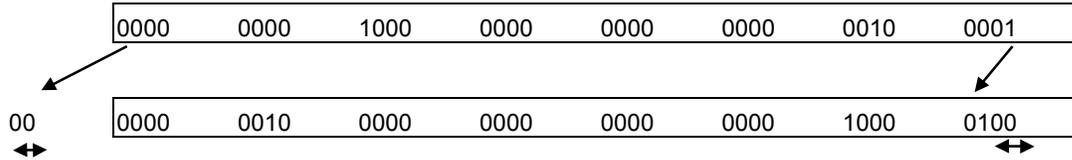
A blank is required before and after the SHL operator.

[Application example]

```
MOV A, #800021H SHL 2 ; (b)
```

[Explanation]

This operator shifts the value "800021H" to the left by 2 bits.



Left-shifted by 2 bits.

0's are inserted.

The value "000084H" is returned.

Therefore, (b) in the above example can also be described as: MOV A, #84H

(1) HIGH operator**[Function]**

Returns the higher 8-bit value of the lowest 16 bits of a term.
A blank is required between the HIGH operator and the term.

[Application example]

```
MOV A, #HIGH 123456H ; (a)
```

[Explanation]

By executing a MOV instruction, this operator returns the higher 8-bit value "34H" of the lower 16 bits of the expression "123456H".

Therefore, (a) in the above example can also be described as: MOV A, #34H

(2) LOW operator**[Function]**

Returns the lower 8-bit value of the lowest 16 bits of a term.
A blank is required between the LOW operator and the term.

[Application example]

```
MOV A, #LOW 123456H ; (b)
```

[Explanation]

By executing a MOV instruction, this operator returns the lower 8-bit value "56H" of the lower 16 bits of the expression "123456H".

Therefore, (b) in the above example can also be described as: MOV A, #56H

(1) HIGHW**[Function]**

Returns the higher 8-bit value of a 32-bit term.

A blank is required between the HIGHW operator and the term.

[Application example]

```
MOVW AX, #HIGHW 12345678H ; (a)
```

[Explanation]

By executing a MOVW instruction, this operator returns the higher 8-bit value "12H" of the 32-bit term "12345678H".

Therefore, (a) in the above example can also be described as: MOVW AX, #12H

(2) LOWW**[Function]**

Returns the lower 16-bit value of a 32-bit term.

A blank is required between the LOWW operator and the term.

[Application example]

```
MOVW AX, #LOWW 12345678H ; (b)
```

[Explanation]

By executing a MOVW instruction, this operator returns the lower 16-bit value "5678H" of the 32-bit term "12345678H".

Therefore, (b) in the above example can also be described as: MOVW AX, #5678H

(1) DATAPOS**[Function]**

Returns the address portion (byte address) of a bit symbol.

[Application example]

```

        SYM    EQU    0FE68H.6

        MOV    A, !DATAPOS SYM    ; (a)

```

[Explanation]

The EQU directive defines the name "SYM" with the value 0FE68H.6.

"DATAPOS SYM" represents "DATAPOS 0FE68H.6", and "0FE68H" is returned.

Therefore, (a) in the above example can also be described as: MOV A, !0FE68H

(2) BITPOS**[Function]**

Returns the bit portion (bit position) of a bit symbol.

[Application example]

```

        SYM    EQU    0FE68H.6

        CLR1   [HL] .BITPOS SYM    ; (b)

```

[Explanation]

The EQU directive defines the name "SYM" with the value 0FE68H.6.

"BITPOS.SYM" represents "BITPOS 0FE68H.6", and "6" is returned.

The CLR1 instruction clears [HL].6 to 0.

(3) MASK**[Function]**

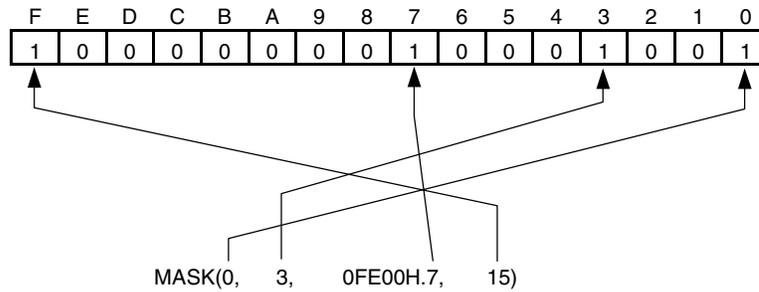
Returns a 16-bit value in which the specified bit position is 1 and all others are set to 0.

[Application example]

```
MOVW AX, #MASK(0, 3, 0FE00H.7, 15)
```

[Explanation]

The MOVW instruction returns the value "8089H".



(1) ()

[Function]

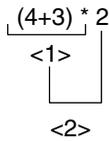
Causes an operation in parentheses to be performed prior to operations outside the parentheses.

This operator is used to change the order of precedence of other operators.

If parentheses are nested at multiple levels, the expression in the innermost parentheses will be calculated first.

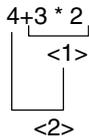
[Application example]

```
MOV  A,  # (4+3) *2
```

[Explanation]

Calculations are performed in the order of expressions <1> and <2> and value “14” is returned as a result.

If parentheses are not used,



Calculations are performed in the order <1> <2> shown above, and the value “10” is returned as a result.

See **Table 2-8 Order of Precedence of Operators**, for the order of precedence of operators.

2.3.2 Restrictions on operations

The operation of an expression is performed by connecting terms with operator(s). Elements that can be described as terms include constants, \$, names, and labels. Each term has a relocation attribute and a symbol attribute.

Depending on the types of relocation attribute and symbol attribute inherent in each term, operators that can work on the term are limited. Therefore, when describing an expression, it is important to pay attention to the relocation attribute and symbol attribute of each of the terms constituting the expression.

(1) Operators and relocation attributes

As previously mentioned, each of the terms that constitute an expression has a relocation attribute and symbol attribute.

Terms can be divided into three types when classified by their relocation attributes: Absolute terms, relocatable terms, and external reference terms.

Types of relocation attributes in operations, the nature of each attribute, and terms applicable to each attribute are shown in **Table 2-9 Types of Relocation Attributes**.

Table 2-9. Types of Relocation Attributes

Type	Nature	Applicable Terms
Absolute term	Term whose value and constant are determined at assembly time	<ul style="list-style-type: none"> • Constants • Labels defined within an absolute segment • \$ indicating the location address defined within an absolute segment • Names defined with constants, the above labels, the above \$, or absolute values
Relocatable term	Term whose value is not determined at assembly time	<ul style="list-style-type: none"> • Labels defined within a relocatable segment • \$ indicating the location address defined within a relocatable segment • Names defined with a relocatable symbol
External reference term ^{Note}	Term that externally references the symbol of another module	<ul style="list-style-type: none"> • Labels defined with the EXTRN directive • Names defined with the EXTBIT directive

Note The following six operators can work on external reference terms: '+', '-', 'HIGH', 'LOW', 'HIGHW', and 'LOWW'. Only one external reference symbol can be described in an expression. In this case, the external reference symbol must be connected with a "+" operator.

Combinations of the type of operator and terms on which each operator can work are shown in **Table 2-10 Combinations of Terms and Operators by Relocation Attribute**.

Table 2-10. Combinations of Terms and Operators by Relocation Attribute (1/2)

Relocation Attribute of Term Type of Operator	X:ABS Y:ABS	X:ABS Y:REL	X:REL Y:ABS	X:REL Y:REL
X + Y	A	R	R	—
X - Y	A	—	R	A ^{Note}
X * Y	A	—	—	—
X / Y	A	—	—	—
X MOD Y	A	—	—	—
X SHL Y	A	—	—	—
X SHR Y	A	—	—	—
X EQ Y	A	—	—	A ^{Note}
X LT Y	A	—	—	A ^{Note}
X LE Y	A	—	—	A ^{Note}
X GT Y	A	—	—	A ^{Note}
X GE Y	A	—	—	A ^{Note}
X NE Y	A	—	—	A ^{Note}
X AND Y	A	—	—	—
X OR Y	A	—	—	—
X XOR Y	A	—	—	—
NOT X	A	A	—	—
+ X	A	A	R	R
- X	A	A	—	—

<Explanation>

ABS: Absolute term

REL: Relocatable term

A: The result of the operation becomes an absolute term.

R: The result of the operation becomes a relocatable term.

—: The operation cannot be performed.

Note The operation can only be performed if X and Y are defined within the same segment, and not relocatable terms on which HIGH, LOW, HIGHW, LOWW, and DATAPOS are operated.

Table 2-10. Combinations of Terms and Operators by Relocation Attribute (2/2)

Relocation Attribute of Term Type of Operator	X:ABS Y:ABS	X:ABS Y:REL	X:REL Y:ABS	X:REL Y:REL
HIGH X	A	A	R ^{Note}	R ^{Note}
LOW X	A	A	R ^{Note}	R ^{Note}
HIGHW X	A	A	R ^{Note}	R ^{Note}
LOWW X	A	A	R ^{Note}	R ^{Note}
MASK (X)	A	A	—	—
DATAPOS X.Y	A	—	—	—
BITPOS X.Y	A	—	—	—
MASK (X.Y)	A	—	—	—
DATAPOS X	A	A	R	R
BITPOS X	A	A	A	A
MASK (X)	A	A	—	—

<Explanation>

ABS: Absolute term

REL: Relocatable term

A: The result of the operation becomes an absolute term.

R: The result of the operation becomes a relocatable term.

—: The operation cannot be performed.

Note The operation can only be performed if X and Y are not relocatable terms on which HIGH, LOW, HIGHW, LOWW, and DATAPOS are operated.

The following six operators can work on external reference terms: '+', '-', 'HIGH', 'LOW', 'HIGHW', and 'LOWW' (however, note that only one external reference term can be described in an expression).

Combinations of the types of operators and external reference terms on which each operator can work are classified according to relocation attributes in **Table 2-11 Combinations of Terms and Operators by Relocation Attribute (External Reference Terms)**.

Table 2-11. Combinations of Terms and Operators by Relocation Attribute (External Reference Terms)

Relocation Attribute of Term Type of Operator	X:ABS Y:EXT	X:EXT Y:ABS	X:REL Y:EXT	X:EXT Y:REL	X:EXT Y:EXT
X + Y	E	E	—	—	—
X – Y	—	E	—	—	—
+ X	A	E	R	E	E
HIGH X	A	E ^{Note 1}	R ^{Note 2}	E ^{Note 1}	E ^{Note 1}
LOW X	A	E ^{Note 1}	R ^{Note 2}	E ^{Note 1}	E ^{Note 1}
HIGHW X	A	E ^{Note 1}	R ^{Note 2}	E ^{Note 1}	E ^{Note 1}
LOWW X	A	E ^{Note 1}	R ^{Note 2}	E ^{Note 1}	E ^{Note 1}
MASK (X)	A	—	—	—	—
DATAPOS X.Y	—	—	—	—	—
BITPOS X.Y	—	—	—	—	—
MASK (X.Y)	—	—	—	—	—
DATAPOS X	A	E	R	E	E
BITPOS X	A	E	A	E	E

<Explanation>

ABS: Absolute term

REL: Relocatable term

A: The result of the operation becomes an absolute term.

E: The result of the operation becomes an external reference term.

R: The result of the operation becomes a relocatable term.

—: The operation cannot be performed.

- Notes**
1. The operation can only be performed if X and Y are not external reference terms on which HIGH, LOW, HIGHW, LOWW, DATAPOS, and BITPOS are operated.
 2. The operation can only be performed if X and Y are not relocatable terms on which HIGH, LOW, HIGHW, LOWW, and DATAPOS are operated.

(2) Operators and symbol attributes

As previously mentioned, each of the terms that constitute an expression has a symbol attribute in addition to a relocation attribute. Terms can be divided into two types when classified by their symbol attributes: NUMBER terms and ADDRESS terms.

Types of symbol attributes in operations and terms applicable to each attribute are shown in **Table 2-12 Types of Symbol Attributes in Operations**.

Table 2-12. Types of Symbol Attributes in Operations

Type of Symbol Attribute	Applicable Terms
NUMBER term	<ul style="list-style-type: none">• Symbols that have NUMBER attribute• Constants
ADDRESS term	<ul style="list-style-type: none">• Symbols that have ADDRESS attribute• \$ indicating the location counter

Combinations of the type of operator and terms on which each operator can work when classified by their symbol attributes are shown in **Table 2-13 Combinations of Terms and Operators by Symbol Attribute**.

Table 2-13. Combinations of Terms and Operators by Symbol Attribute

Symbol Attribute of Term Type of Operator	X:ADDRESS Y:ADDRESS	X:ADDRESS Y:NUMBER	X:NUMBER Y:ADDRESS	X:NUMBER Y:NUMBER
X + Y	—	A	A	N
X - Y	N	A	—	N
X * Y	—	—	—	N
X / Y	—	—	—	N
X MOD Y	—	—	—	N
X SHL Y	—	—	—	N
X SHR Y	—	—	—	N
X EQ Y	N	—	—	N
X LT Y	N	—	—	N
X LE Y	N	—	—	N
X GT Y	N	—	—	N
X GE Y	N	—	—	N
X NE Y	N	—	—	N
X AND Y	—	—	—	N
X OR Y	—	—	—	N
X XOR Y	—	—	—	N
NOT X	—	—	N	N
+ X	A	A	N	N
- X	—	—	N	N
HIGH X	A	A	N	N
LOW X	A	A	N	N
HIGHW X	A	A	N	N
LOWW X	A	A	N	N
DATAPOS X	A	A	N	N
MASK X	N	N	N	N

<Explanation>

ADDRESS: ADDRESS term

NUMBER: NUMBER term

A: The result of the operation becomes an ADDRESS term.

N: The result of the operation becomes a NUMBER term.

—: The operation cannot be performed.

(3) How to check restrictions on the operation

An example of an operation by the relocation attribute and symbol attribute of each term is shown here.

Example BR \$TABLE+5H

Here, assume that "TABLE" is a label defined in a relocatable code segment.

<1> Operator and relocation attribute

Because "TABLE+5H" is "relocatable term+absolute term", this operation is applied to **Table 2-10**

Combinations of Terms and Operators by Relocation Attribute.

Type of operator	X+Y
Relocation attribute of term	X:REL, Y:ABS

From the table, it can be seen that the result is R (namely, a relocatable term).

<2> Operator and symbol attribute

Because "TABLE+5H" is "ADDRESS term+NUMBER term", this operation is applied to **Table 2-13**

Combinations of Terms and Operators by Symbol Attribute.

Type of operator	X+Y
Symbol attribute of term	X:ADDRESS, Y:NUMBER

From the table, it can be seen that the result is A (namely, an ADDRESS term).

2.4 Bit Position Specifier

Bits can be accessed by using the bit position specifier (.).

Bit Position Specifier

Bit Position Specifier

(1) Period (.) (bit position specifier)**[Description format]****Combinations of X (1st Term) and Y (2nd Term)**

X (1st Term)		Y (2nd Term)
General-purpose register	A	Expression (0 to 7)
	X	Expression (0 to 7)
Control register	PSWL	Expression (0 to 7)
	PSWH	Expression (0 to 7)
Special function register	sfr ^{Note}	Expression (0 to 7)
Memory	[DE] ^{Note}	Expression (0 to 7)
	[HL] ^{Note}	Expression (0 to 7)

Note For details on the specific description, see the user's manual of each device.

[Function]

- The bit position specifier specifies a byte address with its 1st term and the position of a bit by its 2nd term. A specific bit can be accessed by this bit position specifier.

[Explanation]

- A bit term refers to an expression that uses a bit position specifier.
- The bit position specifier is not affected by the precedence order of operators. The left side of the bit position specifier is recognized as the 1st term and its right side as the 2nd term.
- The following restrictions apply to the 1st term.
 - <1> An expression with the NUMBER or ADDRESS attribute, an SFR name capable of 8-bit access, or register name (A) can be described.
 - <2> When an absolute expression is described in the 1st term, it must be within the range of 0FE20H to 0FF1FH. However, the range varies according to the CHGSFR control instruction or by specifying an assembler option (-CS).
 - <3> An external reference symbol can be described.
- The following restrictions apply to the 2nd term:
 - <1> The value of an expression must be in the range of 0 to 7. If this value range is exceeded, an error will result.
 - <2> Only an absolute expression with the NUMBER attribute can be described.
 - <3> No external reference symbol can be described.

[Operations and relocation attributes]

- Combinations of the 1st and 2nd terms by relocation attribute are shown in **Table 2-14 Combinations of 1st and 2nd Terms by Relocation Attribute**.

Table 2-14. Combinations of 1st and 2nd Terms by Relocation Attribute

Combination of Terms	X:	ABS	ABS	REL	REL	ABS	EXT	REL	EXT	EXT
	Y:	ABS	REL	ABS	REL	EXT	ABS	EXT	REL	EXT
X.Y		A	—	R	—	—	E	—	—	—

<Explanation>

- | | |
|------------------------------|--|
| ABS: Absolute term | A: The result of the operation becomes an absolute term. |
| REL: Relocatable term | R: The result of the operation becomes a relocatable term. |
| EXT: External reference term | E: The result of the operation becomes an external reference term. |
| | —: The operation cannot be performed. |

[Values of Bit Symbols]

- When a bit symbol is defined by describing a bit term using the bit position specifier in the operand field of the EQU directive, the value that the bit symbol will have is shown in **Table 2-15 Values of Bit Symbols**, below.

Table 2-15. Values of Bit Symbols

Operand Type	Symbol Value
A.bit1 ^{Note 2}	1H.bit1
X.bit1 ^{Note 2}	0H.bit1
PSWL.bit1 ^{Note 2}	1FEH.bit1
PSWH.bit1 ^{Note 2}	1FFH.bit1
sfr ^{Note 1} .bit1 ^{Note 2}	0xxxxxH.bit1 ^{Note 3}
expression.bit1 ^{Note 2}	0xxxxH.bit1 ^{Note 4}

- Notes**
1. For a detailed description, refer to the user's manual of each device.
 2. bit1 = 0 to 7
 3. 0xxxxxH denotes the address of an sfr.
 4. 0xxxxH denotes the value of an expression.

[Application example]

MOV1	CY, 0FFD20H.3	
AND1	CY, A.5	
CLR1	P1.2	
SET1	1+0FFD30H.3	; Equals 0FFD31H.3
SET1	0FFD40H.4+2	; Equals 0FFD40H.6

2.5 Characteristics of Operands

Instructions and directives requiring an operand or operands differ from one type of instruction to another in the size and address range of the required operand value and in the symbol attribute of the operand.

For example, the instruction “MOV r, #byte” functions to transfer the value indicated by “byte” to register “r”. In this case, because r is an 8-bit register, the size of the data “byte” to be transferred must be 8 bits or less.

If an instruction is described as “MOV R0, #100H”, an assembly error occurs, because the size of the 2nd operand “100H” of the instruction exceeds the capacity of the 8-bit register R0.

When describing an operand, therefore, attention must be paid to the following points.

- Is the size of the operand value or its address range suitable for the operand (numerical data, name, or label) of the instruction?
- Is the symbol attribute suitable for the operand (name or label) of the instruction?

2.5.1 Size and address range of operand value

Certain conditions are set for the size and address range of the value of the numerical data, name, or label that can be described as the operand of an instruction.

With instructions, conditions for the size and address range of an operand value are governed by the operand representation format of each instruction. With directives, conditions for the size and address range of an operand value are governed by the type of instructions.

These conditions are shown in **Tables 2-16 Ranges of Operand Values of Instructions** and **2-17 Ranges of Operand Values of Directives**, below.

Table 2-16. Ranges of Operand Values of Instructions

Operand Representation Format	Range of Value	
byte	8-bit value 0H to 0FFH	
word	16-bit value 0H to 0FFFFH	
imm24	24-bit value 0H to 0FFFFFFH	
saddr1	xFE00H to xFEFFH ^{Note 1}	
saddrg1	xFE00H to xFEFDH ^{Note 1}	
saddrp1	Even value of xFE00H to xFEFFH ^{Note 1}	
saddr2 ^{Note 2}	xFD20H to xFDFFH ^{Note 1}	
	xFF00H to xFF1FH ^{Note 1}	
saddrg2	xFD20H to xFDFFH ^{Note 1}	
saddrp2	Even value of xFD20H to xFDFFH ^{Note 1}	
	xFF00H to xFF1FH ^{Note 1}	
sfr	xFF20H to xFFFFH ^{Note 1}	
sfrp	Even value of xFF20H to xFFFFH ^{Note 1}	
addr24	0H to 0FFFFFFH	
addr20 ^{Note 3}	0H to 0FCFFH, 10000H to FFFFFH	
addr16 ^{Note 3}	MOVTBLW	xFE00H to xFEFFH ^{Note 1}
	Other instructions	0H to 0FCFFH
addr16 of MOVTBLW ^{Note 3}	nFE00H to nFEFFH	
addr11	800H to 0FFFH	
addr5	Even value of 40H to 7EH	
bit	3-bit value 0 to 7	
n	3-bit value 0 to 7	
n8 of MOVTBLW, MACW	8-bit (0H to FFH)	
locaddr	0H, 0FH	

- Notes**
1. The range varies depending on the part number of each target device. The default value of x is F. The range of x can be changed by the SFR area change control instruction (CHGSFR).
 2. The range xFF00H to xFF1FH is not available in the case of saddrp2.
 3. Symbols may be odd-numbered addresses.

Table 2-17. Ranges of Operand Values of Directives

Type of Directive	Directive	Range of Values
Segment definition directives	CSEG AT	0H to 0FCFFH, 10000H to FFFFFH ^{Note 1}
	DSEG AT	0H to 0FFFFFFH
	BSEG AT	0H to 0FFFFFFH ^{Note 2}
	ORG	0H to 0FFFFFFH
Symbol definition directives	EQU	24-bit value 0H to 0FFFFFFH
	SET	24-bit value 0H to 0FFFFFFH
Memory initialization and area reservation directives	DB	8-bit value 0H to 0FFH
	DW	16-bit value 0H to 0FFFFH
	DG	24-bit value 0H to 0FFFFFFH
	DS	24-bit value 0H to 0FFFFFFH
Automatic branch instruction selection directives	BR	0H to 0FFEFFH
	CALL	0H to 0FFEFFH
General-purpose register selection directive	RSS	1-bit value 0, 1

- Notes**
1. This shows the default value range. The range can be changed by the SFR area change control instruction (CHGSFR). For details, see **4.8 SFR Area Change Control Instructions**.
 2. 0H to 0FFFFFFH does not include the SFR area.

2.5.2 Size of operands required for instructions

Instructions can be classified into machine instructions and directives. For instructions that require immediate data and symbols as operands, the size of the operand required varies for each instruction.

Therefore, when data in excess of the size of the operand required for the instruction is described, an error occurs. The operations of expressions are carried out with unsigned 32 bits. If the evaluation result exceeds 0FFFFFFH (24 bits), a warning message is output.

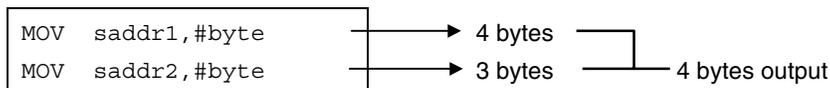
However, when relocatable or external-reference symbols are described in an operand, the values are not determined within the assembler. Instead, the linker determines the values and checks the value range.

In this case as well, as shown in the "Value Appropriateness Check" column of **Table 2-19 Properties of Symbols Describable as Operands of Directives**, a value range check is not performed for some of the operands. Note that only the necessary parts are retrieved and embedded in the object.

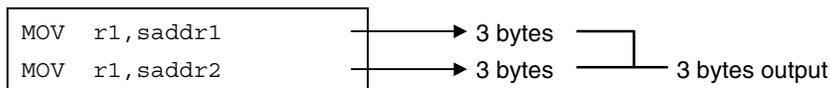
- Cautions about the saddr field

When a mnemonic can reference the SADDR field forward for absolute description, and backward or forward for relocatable description and has both the saddr1 and saddr2 operand description formats (or saddrg1 and saddrg2, to which the discussion of saddr1 and saddr2 below also applies), the assembler outputs the object size of the longest of the two formats saddr1 and saddr2 (if they are the same size, there is no problem).

[Example 1]



[Example 2]



In the case of forward reference of an absolute description, the decision whether the symbol output is saddr1 or saddr2 can be made as soon as pass 1 is finished. When the object code is output at pass 2, the appropriate object code, either saddr1 or saddr2, is output.

At this time, if the object size is different from that determined at pass 1, "00" is output as a nop to complement the difference in object sizes (usually 1 byte), and a warning error message (W714) is output.

[Example]

```

                DESG    AT    ????
SYM:           DB      1
                MOV     SYM, #10H

```

If SYM is saddr1, the assembler outputs XXXXXXXX (4 bytes) for the object code [MOV saddr1, #byte]. If SYM is saddr2, the assembler outputs YYYYYY00 (3 bytes+00) for the object code [MOV saddr2, #byte].

In the case of a relocatable backward or forward reference, it is impossible to determine whether the symbol is saddr1 or saddr2. Therefore, the assembler outputs to the list file (and the object file) the object code of the longest of the two description formats saddr1 and saddr2, which is saddr1.

The linker determines the address for the symbol and whether the symbol is saddr1 or saddr2, and corrects the object code to whichever of saddr1 and saddr2 is appropriate. At this point, if the object size is different from the size determined by the assembler, the difference in object sizes (usually 1 byte) is output as "00" for nop, and a warning error message (W714) is output.

[Example]

```

                DESG
SYM:           DB      1
                MOV     SYM, #10H

```

If the object code is saddr1, the linker does not correct the object code XXXXXXXX output by the assembler. If it is saddr2, the linker corrects the object code by adding "00" to YYYYYY to output YYYYYY00 (4 bytes) for the object code [MOV saddr2, #byte].

- Supporting functions for users of 78K/II, 78K/III source programs

For customers using 78K/IV for 78K/II, 78K/III source programs, a function is available to support stack operation instructions.

When 78K/IV reads "MOV SP, #WORD", which was the description function for 78K/II, 78K/III, a warning message (W713) and the object code "MOVG SP, #imm24" are output.

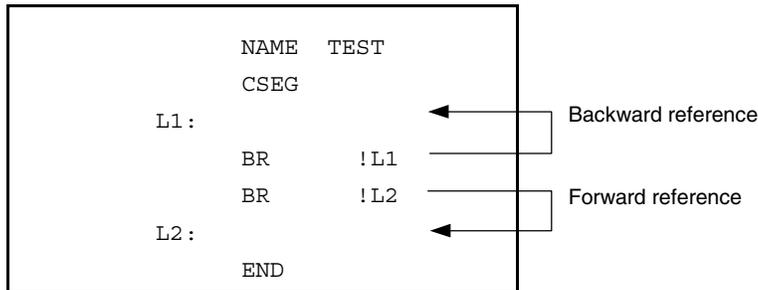
2.5.3 Symbol attributes and relocation attributes of operands

When names, labels, and \$ (which indicates the location counter) are described as instruction operands, they may or may not be describable as operands. This depends on the symbol attributes and relocation attributes (see **2.3.2 Restrictions on operations**) that serve as the terms of their expressions, as well as on the direction of reference in the case of names and labels.

The reference direction for names and labels can be backward reference or forward reference.

- Backward reference ... A name or label referenced as an operand, which is defined in a line above (before) the name or label
- Forward reference ... A name or label referenced as an operand, which is defined in a line below (after) the name or label

[Example]



These symbol attributes and relocation attributes, as well as direction of reference for names and labels, are shown in **Table 2-18 Attributes of Instruction Operands**, and **Table 2-19 Properties of Symbols Describable as Operands of Directives**.

Table 2-18. Attributes of Instruction Operands

	Absolute Expression	Defined by SET, EQU Directive		Relocation Attributes of CSEG/DSEG Segments for Which Labels Exist or Relocation Attributes Specified by EXTRN Directive														SFR Reserved Words	
				saddr ^{*1} / saddrp		saddr2 ^{*1} / saddrp2		saddr3 ^{*1}		fixed / fixeda		callt0		Other ^{*2}		None ^{*1,3}			
Reference pattern	–	Backward	Forward	Backward	Forward	Backward	Forward	Backward	Forward	Backward	Forward	Backward	Forward	Backward	Forward	Backward	Forward		
Description format																			
byte	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	x
word	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	x
imm24	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	x
saddr1	○	○	○	○	○	x	x	○	○	○	○	○	○	○	○	○	○	○	x
saddrg1	○	○	○	○ ^{*5}	○ ^{*5}	x	x	○	○	○	○	○	○	○	○	○	○	○	x
saddrp1	○	○	○	○ ^{*5}	○ ^{*5}	–	–	○	○	○	○	○	○	○	○	○	○	○	x
saddr2 ^{*6}	^{*7}	○	○	○ ^{*9}	x	x	○ ^{*5}	○ ^{*5,9}	○ ^{*10}	○ ^{*10}	–	–	–	–	–	–	–	–	
	^{*8}	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	○ ^{*11}	
saddrg2	○	○	○ ^{*9}	x	x	○ ^{*5}	○ ^{*5}	○ ^{*10}	○ ^{*10}	–	–	–	–	–	–	–	–	–	
saddrp2	^{*7}	○	○	○ ^{*9}	x	x	○	○	○ ^{*10}	○ ^{*10}	–	–	–	–	–	–	–	–	
	^{*8}	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	○ ^{*12}	
sfr	○ ^{*13}	○ ^{*13}	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	○ ^{*14}
sfrp	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	○ ^{*3}
addr24	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	x
addr20	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	x
addr16	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	x
addr16 of MOVTLW	○	○	○	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
addr11	○	○	○	x	x	x	x	x	x	○	○	x	x	x	x	x	x	x	x
addr5	○	○	○	x	x	x	x	x	x	x	x	○	○	x	x	x	x	x	x
bit	○	○	○	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
n	○	○	○	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
n8 of MOVTLW, MACW	○	○	○	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
locaddr	○	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

<Explanation>

- Forward: This means forward reference.
Backward: This means backward reference.
o: This means that description is possible.
x: This means an error.
-: This means that description is not possible.

Notes (*)

1. This is performed by specification of relocation attributes using the extern directive.
2. Relocation attributes other than those in the columns at, gram, unit, unitp, base, etc.
3. Only sfr reserved words for which 16-bit access is possible
4. Relocation attributes are not specified by the "extrn sym" format.
5. Symbols can be odd-number address.
6. In the case of saddr2, this does not include nFF00H to nFF1FH
7. nFD20H to nFDFFH
8. nFF00H to nFF1FH
9. The assembler may append a nop "00" to the object code. For details, see **2.5.2 Size of operands required for instructions**.
10. The assembler outputs the object code for saddr1/saddrg1. In some cases, the linker may append a nop "00". For details, see **2.5.2 Size of operands required for instructions**.
11. 8-bit accessible sfr-reserved words for the area accessible by saddr2 in the sfr field.
12. 16-bit accessible sfr-reserved words for the area accessible by saddr2 in the sfr field.
13. Only absolute expressions are possible in the external access area range.
14. Only sfr reserved words for which 8-bit access is possible

Table 2-19. Properties of Described Symbols as Operands of Directives

Symbol Attributes		NUMBER		ADDRESS, SADDR1, SADDR2						BIT						Value Appropriateness Check
Relocation Attributes		Absolute Terms		Absolute Terms		Relocatable Terms		External Reference Terms		Absolute Terms		Relocatable Terms		External Reference Terms		
Reference direction	Directive	Backward	Forward	Backward	Forward	Backward	Forward	Backward	Forward	Backward	Forward	Backward	Forward	Backward	Forward	
ORG		○ ^{Note 1}	—	—	—	—	—	—	—	—	—	—	—	—	—	⊙
EQU ^{Note 2}		○	—	○	—	○ ^{Note 3}	—	—	—	○	—	○ ^{Note 3}	—	—	—	○
SET		○ ^{Note 1}	—	—	—	—	—	—	—	—	—	—	—	—	—	—
DB	Size	○ ^{Note 1}	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	Initial value	○	○	○	○	○	○	○	○	—	—	—	—	—	—	⊙
DW	Size	○ ^{Note 1}	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	Initial value	○	○	○	○	○	○	○	○	—	—	—	—	—	—	x (16)
DG	Size	○ ^{Note 1}	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	Initial value	○	○	○	○	○	○	○	○	—	—	—	—	—	—	x (24)
DS		○	—	—	—	—	—	—	—	—	—	—	—	—	—	—
BR/CALL		○	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RSS		○ ^{Note 1}	—	—	—	—	—	—	—	—	—	—	—	—	—	⊙

○: Description possible —: Description impossible

<Explanation>

The "Value Appropriateness Check" column refers to a check of the appropriateness of the value derived by the assembler if it is an absolute expression, and by the linker if it is a relocatable or external reference expression.

- ⊙: Check is carried out according to the value range.
- : Check is carried out according to the value range of saddr1.bit/saddr2.bit/external access field.bit, for an absolute BIT in the assembler only
- x (16): Lower 16 bits are embedded in the object as a result of calculation.
- x (24): Lower 24 bits are embedded in the object as a result of calculation.
- : No value range

- Notes**
1. Only an absolute expression can be described.
 2. An error will result if an expression that includes one of the following 8 patterns and that produces a result that is affected by optimization is described.
The SADDR1 and SADDR2 attributes are included in these ADDRESS attributes.
 - ADDRESS attribute - ADDRESS attribute
 - ADDRESS attribute relational operator ADDRESS attribute
 - HIGH absolute ADDRESS attribute
 - LOW absolute ADDRESS attribute
 - HIGHW absolute ADDRESS attribute
 - LOWW absolute ADDRESS attribute
 - DATAPOS absolute ADDRESS attribute
 - MASK absolute ADDRESS attribute
 3. A term created by the HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS/MASK operator which has a relocatable term is not allowed.

CHAPTER 3 DIRECTIVES

This chapter explains the directives. Directives are instructions that direct all types of instructions necessary for the RA78K4 to perform a series of processes.

3.1 Overview of Directives

Instructions are translated into object codes (machine language) as a result of assembling, but directives are not converted into object codes in principle. Directives have the following main functions.

- To facilitate description of source programs
- To initialize memory and reserve memory areas
- To provide the information required for assemblers and linkers to perform their intended processing

Table 3-1 List of Directives shows the types of directives.

Table 3-1. List of Directives

No.	Type of Directive	Directive
1	Segment definition directives	CSEG, DSEG, BSEG, ORG
2	Symbol definition directives	EQU, SET
3	Memory initialization/area reservation directives	DB, DW, DG, DS, DBIT
4	Linkage directives	PUBLIC, EXTRN, EXTBIT
5	Object module name declaration directive	NAME
6	Automatic selection directive	BR, CALL
7	General-purpose register selection directive	RSS
8	Macro directives	MACRO, LOCAL, REPT, IRP, EXITM, ENDM
9	Assembly termination directive	END

The following sections explain the details of each directive.

In the description format of each directive, “[]” indicates that the parameter in square brackets may be omitted from specification, and “...” indicates the repetition of description in the same format.

3.2 Segment Definition Directives

A source module must be described in units of segments.

Segment definition directives are used to define these segments. Segments are divided into the following four types.

- Code segments
- Data segments
- Bit segments
- Absolute segments

The type of segment determines the address range in memory in which each segment will be located.

Table 3-2 Segment Definition Methods and Memory Address Location shows the method of defining each segment and the memory address at which each segment is located.

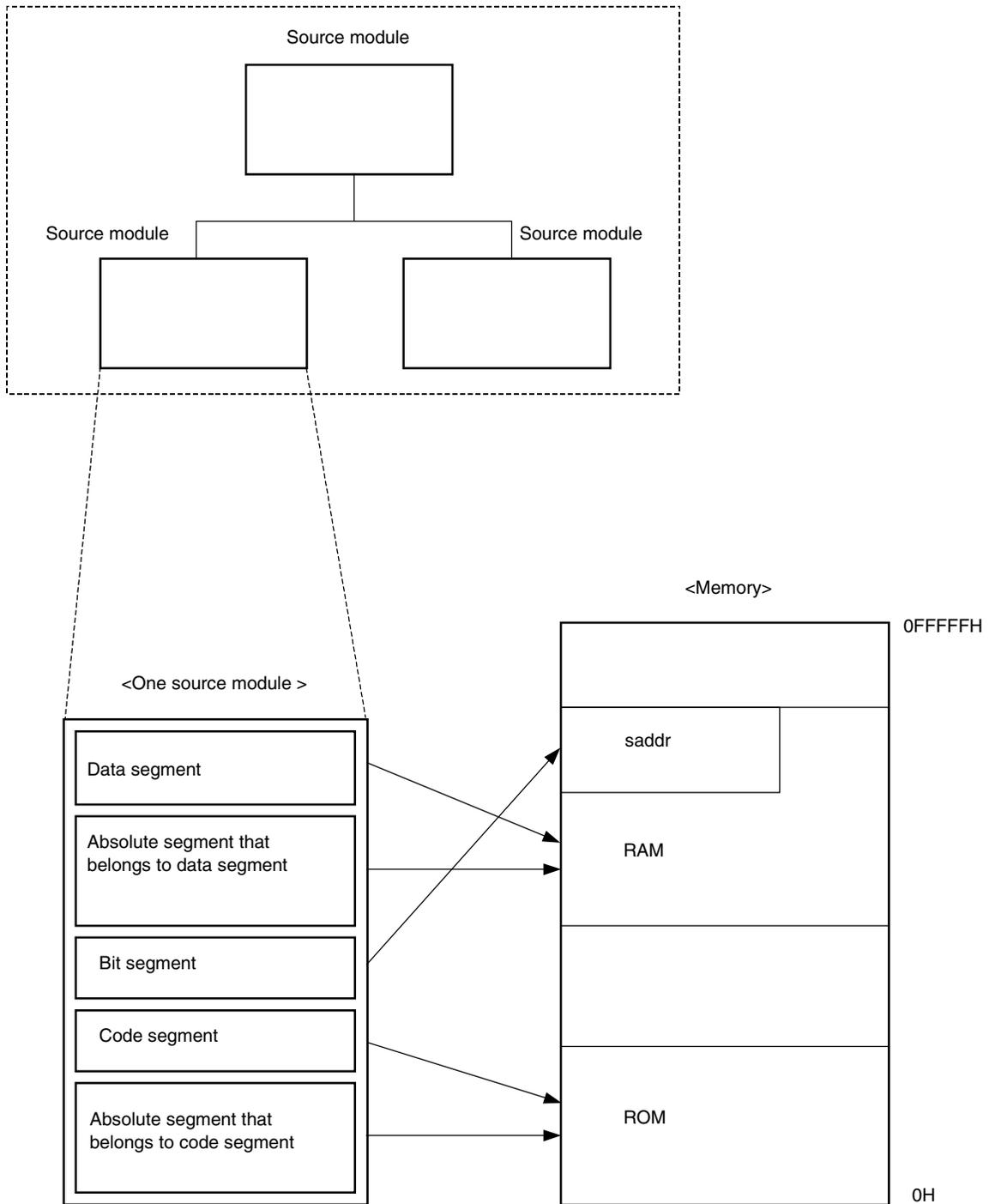
Table 3-2. Segment Definition Methods and Memory Address Location

Type of Segment	Method of Definition	Memory Address at Which Each Segment Is Located
Code segment	CSEG directive	Within the internal or external ROM address
Data segment	DSEG directive	Within the internal or external RAM address
Bit segment	BSEG directive	Within the saddr area in the internal RAM
Absolute segment	Specifies location address (AT location address) for relocation attribute with CSEG, DSEG, or BSEG directive	Specified address

To determine the memory location of a segment, describe the segment as an absolute segment. An area in the data segment must be reserved for the stack area, in which the stack pointer must be set.

An example of segment location is shown in **Figure 3-1 Memory Location of Segments**.

Figure 3-1. Memory Location of Segments



CSEG

code segment

CSEG

(1) CSEG (code segment)

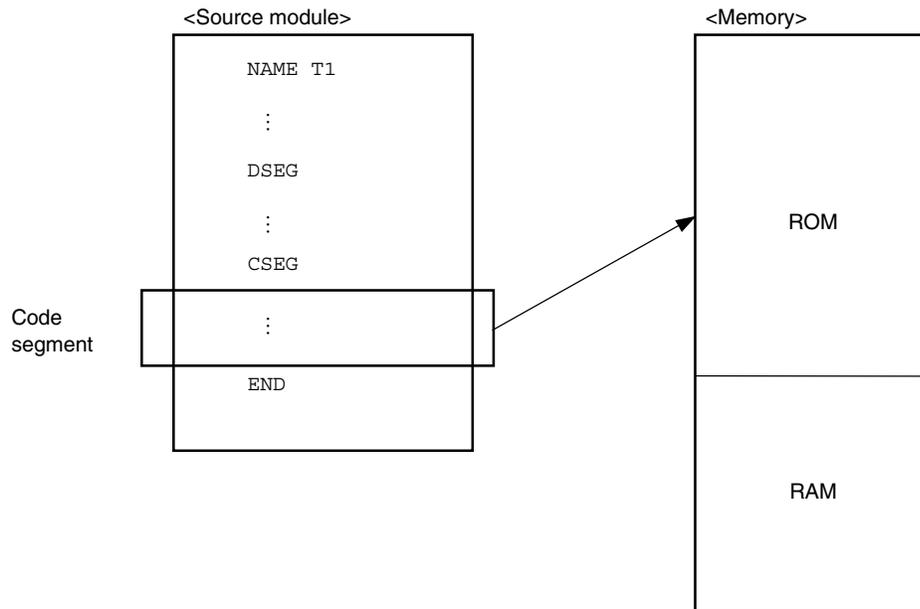
[Description format]

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[segment-name]	CSEG	[relocation-attribute]	[;comment]

[Function]

- The CSEG directive indicates to the assembler the start of a code segment.
- All instructions described following the CSEG directive belong to the code segment until a segment definition directive (CSEG, DSEG, BSEG, or ORG) or the END directive appears, and finally those instructions are located within a ROM address after being converted into machine language.

Figure 3-2. Relocation of Code Segment



[Use]

- The CSEG directive is used to describe instructions, and the DB, DW directives, etc. in the code segment defined by the CSEG directive.
(However, to relocate the code segment from a fixed address, "AT absolute-expression" must be described as its relocation attribute in the operand field.)
- Description of one functional unit such as a subroutine should be defined as a single code segment. If the unit is relatively large or if the subroutine is highly versatile (i.e. can be utilized for development of other programs), the subroutine should be defined as a single module.

CSEG

code segment

CSEG

[Explanation]

- The start address of a code segment can be specified with the ORG directive. It can also be specified by describing the relocation attribute “AT absolute-expression”.
- A relocation attribute defines a range of location addresses for a code segment. Relocation attributes are shown in **Table 3-3 Relocation Attributes of CSEG**.

Table 3-3. Relocation Attributes of CSEG

Relocation Attribute	Description Format	Explanation
CALLT0	CALLT0	Tells the assembler to locate the specified segment so that the start address of the segment becomes a multiple of 2 within the address range 0040H to 007FH. Specify this relocation attribute for a code segment that defines the entry address of a subroutine to be called with the 1-byte instruction "CALLT".
FIXED	FIXED	Tells the assembler to locate the beginning of the specified segment within the address range 0800H to 0FFFH. Specify this relocation attribute for a code segment that defines a subroutine to be called with the 2-byte instruction "CALLF".
FIXEDA	FIXEDA	Tells the assembler to locate the beginning of the specified segment within the address range 0800H to 0FFFH, and the end at 0FCFFH ^{Note} . Specify this relocation attribute for a code segment that defines a subroutine to be called with the 2-byte instruction "CALLF".
AT	AT Absolute-expression	Tells the assembler to locate the specified segment to the absolute address (within 0000H to 0FCFFH or 10000H to 0FFFFFFH) ^{Note} .
UNIT	UNIT	Tells the assembler to locate the specified segment to any address (within 0080H to 0FCFFH or 10000H to 0FFFFFFH) ^{Note} .
UNITP	UNITP	Tells the assembler to locate the specified segment to any address, so that the start of the address may be an even number (within 0080H to 0FCFFH or 10000H to 0FFFFFFH) ^{Note} .
BASE	BASE	Tells the assembler to locate the specified segment to an address within 80H to 0FCFFH ^{Note} .
PAGE	PAGE	Tells the assembler to locate the specified segment to an address within xxx00H to xxxFFH (no higher than 0FFFFFFH).
PAGE64K	PAGE64K	Tells the assembler to locate the specified segment so that it may not straddle the 64K boundary (within 0H to 0FCFFH and 10000H to FFFFFH) ^{Note} .

Note This area may be changed by the SFR area change control instruction (CHGSFR).

- If no relocation attribute is specified for the code segment, the assembler will assume that “UNIT” has been specified.
- If a relocation attribute other than those listed in **Table 3-3 Relocation Attributes of CSEG** is specified, the assembler will output an error message and assume that “UNIT” has been specified. An error will result if the size of each code segment exceeds that of the area specified by its relocation attribute.
- If the absolute expression specified with the relocation attribute “AT” is illegal, the assembler will output an error message and continue processing by assuming the value of the expression to be “0”.

CSEG

code segment

CSEG

The code segment can be named by describing a segment name in the symbol field of the CSEG directive. If no segment name is specified for a code segment, the assembler will automatically give a default segment name to the code segment. The default segment names of the code segments are shown in **Table 3-4 Default Segment Names of CSEG**.

Table 3-4. Default Segment Names of CSEG

Relocation Attribute	Default Segment Name
CALLT0	?CSEGT0
FIXED	?CSEGFY
FIXEDA	?CSEGFYA
BASE	?CSEGB
PAGE	?CSEGP
PAGE64K	?CSEGP64
UNIT (or omitted)	?CSEG
UNITP	?CSEGUP
AT	Segment name cannot be omitted. If the segment name is omitted, it is assumed that the relocation attribute is UNIT and the segment name becomes ?CSEG.

- An error will result if the segment name is omitted when the relocation attribute is AT.
- If two or more code segments have the same relocation attribute (except AT), these code segments may have the same segment name. These same-named code segments are processed as a single code segment within the assembler.
An error will result if the same-named segments differ in their relocation attributes. Therefore, the number of the same-named segments for each relocation attribute is one.
- The same-named code segments in two or more different modules are combined into a single code segment at linkage.
- No segment name can be referenced as a symbol.
- The total number of segments that can be output by the assembler is up to 255 different name segments, including those defined with the ORG directive. The same-named segments are counted as one.
- The maximum number of characters recognizable as a segment name is 8.
- Segment names are case sensitive.

CSEG

code segment

CSEG

[Application examples]

```
      NAME      SAMP1
C1    CSEG                                ; (1)

C2    CSEG      CALLT0                      ; (2)

      CSEG      FIXED                       ; (3)
C1    CSEG      CALLT0                      ; (4)

      CSEG                                ; (5)

      END
```

<Explanation>

- (1) The assembler interprets the segment name as "C1", and the relocation attribute as "UNIT".
- (2) The assembler interprets the segment name as "C2", and the relocation attribute as "CALLT0".
- (3) The assembler interprets the segment name as "?CSEGFx", and the relocation attribute as "FIXED".
- (4) Because the segment name "C1" was defined as the relocation attribute "UNIT" in (1), an error occurs.
- (5) The assembler interprets the segment name as "?CSEG", and the relocation attribute as "UNIT".

DSEG

data segment

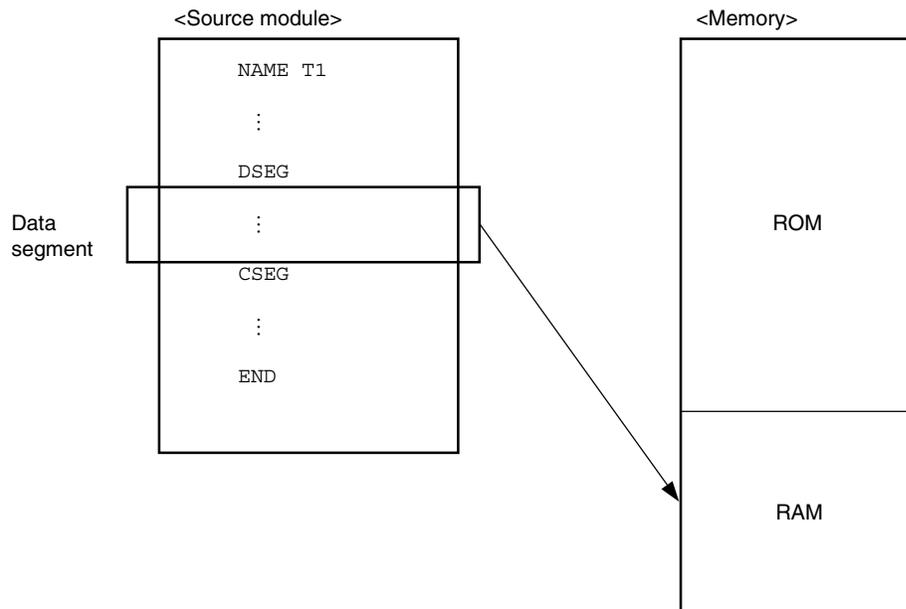
DSEG

(2) DSEG (data segment)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[segment-name]	DSEG	[relocation-attribute]	[:comment]

[Function]

- The DSEG directive indicates to the assembler the start of a data segment.
- A memory defined by the DS directive following the DSEG directive belongs to the data segment until a segment definition directive (CSEG, DSEG, BSEG, or ORG) or the END directive appears, and finally it is reserved within the RAM address.

Figure 3-3. Relocation of Data Segment**[Use]**

- The DS directive is mainly described in data segments defined by the DSEG directive. Data segments are located within the RAM area. Therefore, no instructions can be described in any data segment.
- In a data segment, a RAM work area used in a program is reserved by the DS directive and a label is attached to each work area. Use this label when describing a source program.

Each area reserved as a data segment is located by the linker so that it does not overlap with any other work areas on the RAM (stack area, general-purpose register area, and work areas defined by other modules).

DSEG

data segment

DSEG

[Explanation]

- The start address of a data segment can be specified with the ORG directive. It can also be specified by describing the relocation attribute “AT” followed by an absolute expression in the operand field of the DSEG directive.
- A relocation attribute defines a range of location addresses for a data segment. The relocation attributes available for data segments are shown in **Table 3-5 Relocation Attributes of DSEG**.

Table 3-5. Relocation Attributes of DSEG

Relocation Attribute	Description Format	Explanation
SADDR	SADDR	Tells the assembler to locate the specified segment in the saddr1 area (saddr1 area: 0FE00H to 0FEFFH ^{Note 1}).
SADDR2	SADDR2	Tells the assembler to locate the specified segment in the saddr2 area (saddr2 area: 0FD20H to 0FDFFH ^{Notes 1, 2}).
SADDRP	SADDRP	Tells the assembler to locate the specified segment from an even-numbered address of the saddr1 area (saddr1 area: 0FE00H to 0FEFFH ^{Note 1}).
SADDRP2	SADDRP2	Tells the assembler to locate the specified segment from an even-numbered address of the saddr2 area (saddr2 area: 0FD20H to 0FDFFH ^{Notes 1, 2}).
SADDRA	SADDRA	Tells the assembler to locate the specified segment in an optionally specified area of the saddr area (saddr area: 0FD20H to 0FEFFH (saddr1/saddr2 areas) ^{Notes 1, 2}).
AT	AT absolute-expression	Tells the assembler to locate the specified segment at an absolute address.
UNIT	UNIT or no specification	Tells the assembler to locate the specified segment at an optionally selected location (within the memory area name "RAM" ^{Note 1}).
UNITP	UNITP	Tells the assembler to locate the specified segment at an optionally selected location from an even-numbered address (within the memory area name "RAM" ^{Note 1}).
DTABLE	DTABLE	Tells the assembler to locate the specified segment within the macro service control area (macro service control area: 0FE00H to 0FEFFH ^{Notes 1, 2}).
DTABLEP	DTABLEP	Tells the assembler to locate the specified segment within the macro service control area from an even-numbered address (macro service control area: 0FE00H to 0FEFFH ^{Notes 1, 2}).
LRAM	LRAM	Tells the assembler to locate the specified segment within the peripheral RAM area (in the low-speed RAM). ^{Note 1}
GRAM	GRAM	Tells the assembler to locate the specified segment within the general static RAM area (in the high-speed RAM)(general static RAM area: 0FD00H to 0FEFFH).
PAGE	PAGE	Tells the assembler to locate the specified segment at an optionally selected location from XXXX00H to XXXXFFH (within 0FFFFH).
PAGE64K	PAGE64K	Tells the assembler to locate the specified segment so that it does not straddle the 64K boundary (0H to 0FCFFH and 10000H to FFFFFH ^{Note 2}).

- Notes**
1. The address may vary depending on the type of device for which the program is written.
 2. This shows the default range. The range can be changed by the SFR area change control instruction (CHGSFR).

DSEG

data segment

DSEG

- If no relocation attribute is specified for the data segment, the assembler will assume that “UNIT” has been specified.
- If a relocation attribute other than those listed in **Table 3-5 Relocation Attributes of DSEG** is specified, the assembler will output an error message and assume that “UNIT” has been specified. An error will result if the size of each data segment exceeds that of the area specified by its relocation attribute.
- If the absolute expression specified with the relocation attribute “AT” is illegal, the assembler will output an error message and continue processing by assuming the value of the expression to be “0”.
- By describing a segment name in the symbol field of the DSEG directive, the data segment can be named. If no segment name is specified for a data segment, the assembler automatically gives a default segment name. The default segment names of the data segments are shown in **Table 3-6 Default Segment Names of DSEG**.

Table 3-6. Default Segment Names of DSEG

Relocation Attribute	Default Segment Name
SADDR	?DSEGS
SADDRP	?DSEGSP
SADDR2	?DSEGS2
SADDRP2	?DSEGSP2
SADDRA	?DSEGA
UNIT (or no specification)	?DSEG
UNITP	?DSEGUP
DTABLE	?DSEGDT
DTABLEP	?DSEGDTTP
PAGE	?DSEGP
PAGE64K	?DSEGP64
LRAM	?DSEGL
GRAM	?DSEGG
AT	Segment name cannot be omitted. If the segment name is omitted, it is assumed that the relocation attribute is UNIT and the segment name becomes ?DSEG.

- If two or more data segments have the same relocation attribute (except AT), these data segments may have the same segment name. These segments are processed as a single data segment within the assembler.
- If the relocation attribute is SADDRP, the specified segment is located so that the address immediately after the DSEG directive is described becomes a multiple of 2.
- An error occurs if the same-named segments differ in their relocation attributes. Therefore, the number of the same-named segments for each relocation attribute is one.
- The same-named data segments in two or more different modules are combined into a single data segment at linkage time.
- No segment name can be referenced as a symbol.
- The total number of segments that can be output by the assembler is up to 255 different-name segments including those defined with the ORG directive. The same-named segments are counted as one.
- The maximum number of characters recognizable as a segment name is 8.
- Segment names are case sensitive.

DSEG

data segment

DSEG

[Application examples]

```

NAME      SAMP1
LOCATION   0H
DSEG                                           ; (1)
WORK1: DS      1
WORK2: DS      2
CSEG
MOV      A, !WORK1                            ; (2)
MOV      A, WORK1                             ; (3)
MOVW    DE, #WORK2                            ; (4)
MOVW    AX, [DE]
MOVW    AX, WORK2                             ; (5)
END

```

<Explanation>

- (1) The start of a data segment is defined with the DSEG directive. Because its relocation attribute is omitted, "UNIT" is assumed. The default segment name is "?DSEG".
- (2) This description corresponds to "MOV A, !addr16".
- (3) This description corresponds to "MOV A, saddr". Relocatable label "WORK1" cannot be described as "saddr". Therefore, an error occurs as a result of this description.
- (4) This description corresponds to "MOVW rp, #word".
- (5) This description corresponds to "MOVW AX, saddrp". Relocatable label "WORK2" cannot be described as "saddrp". Therefore, an error occurs as a result of this description.

(3) BSEG (bit segment)

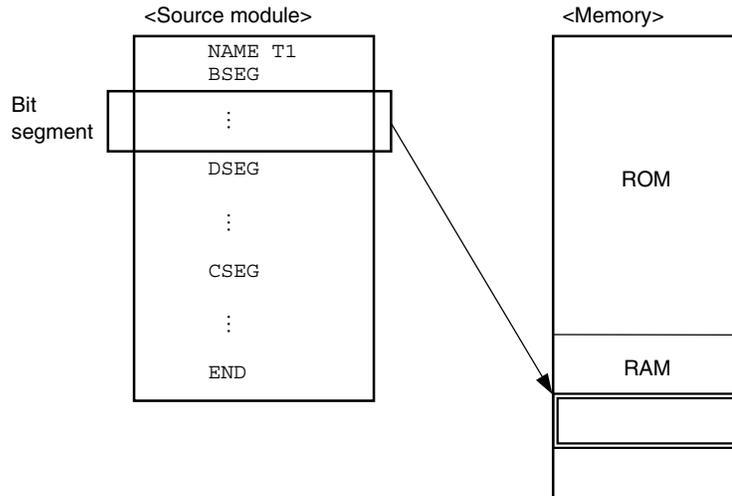
[Description format]

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[segment-name]	BSEG	[relocation-attribute]	[:comment]

[Function]

- The BSEG directive indicates to the assembler the start of a bit segment.
- A bit segment is a segment that defines the RAM addresses to be used in the source module.
- A memory area that is defined by the DBIT directive following the BSEG directive belongs to the bit segment until a segment definition directive (CSEG, DSEG, or BSEG) or the END directive appears.

Figure 3-4. Relocation of Bit Segment



[Use]

- Describe the DBIT directive in the bit segment defined by the BSEG directive (see **Application Example**).
- No instructions can be described in any bit segment.

BSEG

bit segment

BSEG

[Explanation]

- The start address of a bit segment can be specified by describing “AT absolute-expression” in the relocation attribute field.
- A relocation attribute defines a range of location addresses for a bit segment. Relocation attributes available for bit segments are shown in **Table 3-7 Relocation Attributes of BSEG**.

Table 3-7. Relocation Attributes of BSEG

Relocation Attribute	Description Format	Explanation
AT	AT absolute-expression	Tells the assembler to locate the starting address of the specified segment in the 0th bit of an absolute address. Specification in bit units is prohibited (0H to 0FFFFFFH other than the addresses for SFR area).
SADDR	SADDR	Tells the assembler to locate the specified segment in any location in the saddr1 area (saddr1 area: 0FE00H to 0FEFFH ^{Note 1})
SADDR2	SADDR2	Tells the assembler to locate the specified segment in any location in the saddr2 area (saddr2 area: 0FD20H to 0FDFFH ^{Notes 1, 2})
SADDRA	SADDRA	Tells the assembler to locate the specified segment in any location in the saddr area (saddr area: 0FD20H to 0FEFFH ^{Notes 1, 2}).
UNIT	UNIT (or no specification)	Tells the assembler to locate the specified segment in any location in the saddr1 area (saddr1 area: 0FE00H to 0FEFFH ^{Note 1}).
GRAM	GRAM	Tells the assembler to locate the specified segment within the general static RAM area (internal high-speed RAM: 0FD00H to 0FEFFH ^{Note 1}).
ARAM	ARAM	Tells the assembler to locate the specified segment in any location of the entire space (0H to 0FFFFFFH other than the addresses for SFR area).

- Notes**
1. The address may vary depending on the part number of each target device for which the program is written.
 2. This shows the default range. The range can be changed by the SFR area change control instruction (CHGSFR).

- If no relocation attribute is specified for the bit segment, the assembler assumes that “UNIT” is specified.
- If a relocation attribute other than those listed in **Table 3-7 Relocation Attributes of BSEG** is specified, the assembler outputs an error message and assumes that “UNIT” is specified. An error occurs if the size of each bit segment exceeds that of the area specified by its relocation attribute.
- In both the assembler and the linker, the location counter in a bit segment is displayed in the form “00xxxx.b” (The byte address is hexadecimal 6 digits and the bit position is hexadecimal 1 digit (0 to 7)).

With absolute bit segment

Byte address	0	1	2	3	4	5	6	7	Bit position
0FE20H	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	Location counter (1) 0FE20H.0 (9) 0FE21H.0 (2) 0FE20H.1 (10) 0FE21H.1 (3) 0FE20H.2 (11) 0FE21H.2 (4) 0FE20H.3 (12) 0FE21H.3 (5) 0FE20H.4 (13) 0FE21H.4 (6) 0FE20H.5 (14) 0FE21H.5 (7) 0FE20H.6 (15) 0FE21H.6 (8) 0FE20H.7 (16) 0FE21H.7
0FE21H	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	

With relocatable bit segment

Byte address	0	1	2	3	4	5	6	7	Bit position
0H	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	Location counter (1) 0H.0 (9) 1H.0 (2) 0H.1 (10) 1H.1 (3) 0H.2 (11) 1H.2 (4) 0H.3 (12) 1H.3 (5) 0H.4 (13) 1H.4 (6) 0H.5 (14) 1H.5 (7) 0H.6 (15) 1H.6 (8) 0H.7 (16) 1H.7
1H	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	

Remark Within a relocatable bit segment, the byte address specifies an offset value in byte units from the beginning of the segment.

A bit offset from the beginning of an area where a bit is defined is displayed and output in a symbol table output by the object converter.

Symbol Value	Bit Offset
00FE20H.0	0000
00FE20H.1	0001
00FE20H.2	0002
⋮	⋮
00FE20H.7	0007
00FE21H.0	0008
00FE21H.1	0009
⋮	⋮
00FE80H.0	0300
⋮	⋮

BSEG

bit segment

BSEG

- If the absolute expression specified with the relocation attribute “AT” is illegal, the assembler outputs an error message and continues processing while assuming the value of the expression to be “0”.
- By describing a segment name in the symbol field of the BSEG directive, the bit segment can be named. If no segment name is specified for a bit segment, the assembler automatically gives a default segment name. The following table shows the default segment names.

Table 3-8. Default Segment Names of BSEG

Relocation Attribute	Default Segment Name
UNIT (or no specification)	?BSEG
UNITP	?BSEGUP
AT	Segment name cannot be omitted. If the segment name is omitted, it is assumed that the relocation attribute is UNIT and the segment name becomes ?BSEG.
SADDR	?BSEGS
SADDRP	?BSEGSP
SADDR2	?BSEGS2
SADDRP2	?BSEGSP2
SADDRA	?BSEGSA
GRAM	?BSEGG
ARAM	?BSEGA

- If the relocation attribute is “UNIT”, two or more data segments can have the same segment name (except AT). These segments are processed as a single segment within the assembler. Therefore, the number of same-named segments for each relocation attribute is one.
- The same-named bit segments in two or more different modules will be combined into a single bit segment at linkage.
- No segment name can be referenced as a symbol.
- The only instructions that can be described in the bit segments are the DBIT, EQU, SET, PUBLIC, EXTBIT, EXTRN, MACRO, REPT, IRP, ENDM directive, macro definition and macro reference. Description of instructions other than these causes in an error.
- The total number of segments that the assembler outputs is up to 255 different-name segments, with segments defined by the ORG directive. The segments having the same name are counted as one.
- The maximum number of characters recognizable as a segment name is 8.
- Segment names are case sensitive.

BSEG

bit segment

BSEG

[Application examples]

```

NAME      SAMP1

FLAG      EQU      0FE20H
FLAG0     EQU      FLAG.0      ; (1)
FLAG1     EQU      FLAG.1      ; (1)

BSEG                               ; (2)
FLAG2     DBIT

CSEG
MOV1      CY, FLAG0              ; (3)
MOV1      CY, FLAG2              ; (4)

END

```

<Explanation>

(1) Bit addresses (bits 0 and 1 of 0FE20H) are defined with consideration given to byte address boundaries.

(2) A bit segment is defined with the BSEG directive.

Because its relocation attribute is omitted, the relocation attribute "UNIT" and the segment name "?BSEG" are assumed. In each bit segment, a bit work area is defined for each bit with the DBIT directive. A bit segment should be described at the early part of the module body. Bit address FLAG2 defined within the bit segment is located without considering the byte address boundary.

(3) This description can be replaced with "MOV1 CY, FLAG.0". This FLAG indicates a byte address.

(4) In this description, no consideration is given to byte address boundaries.

ORG

origin

ORG

(4) ORG (origin)

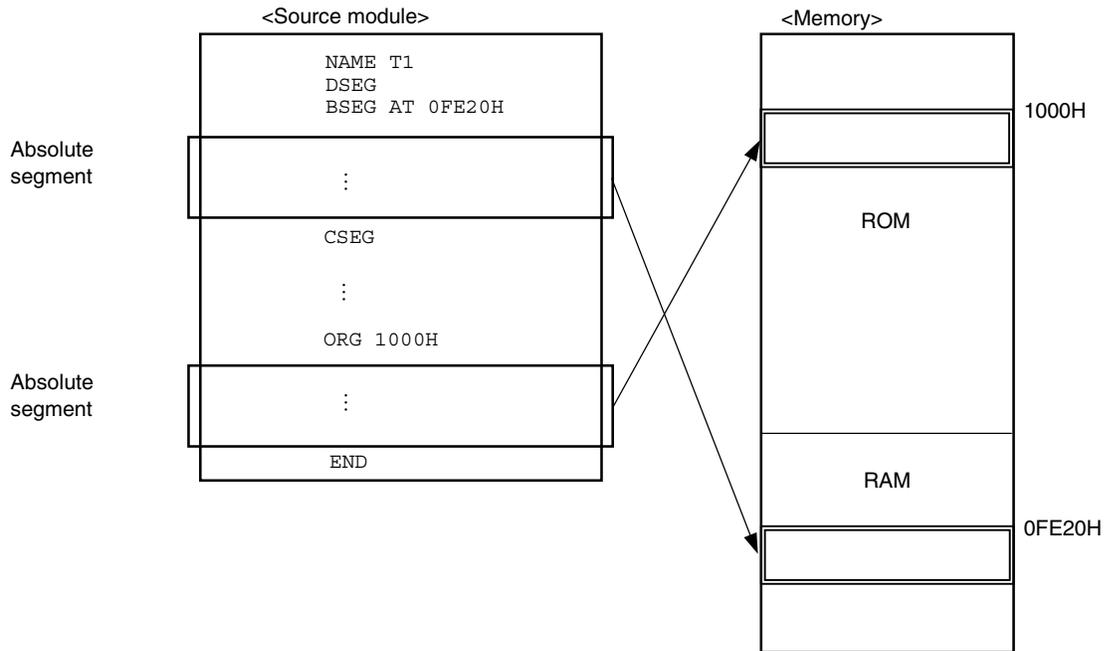
[Description format]

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[segment name]	ORG	Absolute expression	[:comment]

[Function]

- The ORG directive sets the value of the expression specified by its operand of the location counter.
- After the ORG directive, described instructions or reserved memory area belong to an absolute segment until a segment definition directive (CSEG, DSEG, BSEG, or ORG) or the END directive appears, and they are located from the address specified by an operand.

Figure 3-5. Location of Absolute Segment



[Use]

- Specify the ORG directive to locate a code segment or data segment from a specific address.

ORG

origin

ORG

[Explanation]

- The absolute segment defined with the ORG directive belongs to the code segment or data segment defined with the CSEG or DSEG directive immediately before this ORG directive.
No instructions can be described within an absolute segment that belongs to a data segment.
An absolute segment that belongs to a bit segment cannot be described with the ORG directive.
- A code segment or data segment defined with the ORG directive is interpreted as a code segment or data segment of the relocation attribute "AT".
- By describing a segment name in the symbol field of the ORG directive, the absolute segment can be named. The maximum number of characters that can be recognized as a segment name is 8.
- If no segment name is specified for an absolute segment, the assembler will automatically assign the default segment name "?Axxxxxx", where "xxxxxx" indicates the six-digit hexadecimal start address (000000 to FFFFFFFF) of the segment specified.
- If neither CSEG nor DSEG directive has been described before the ORG directive, the absolute segment defined by the ORG directive is interpreted as an absolute segment in a code segment.
- If a name or label is described as the operand of the ORG directive, the name or label must be an absolute term that has already been defined in the source module.
- No segment name can be referenced as a symbol.
- The total number of segments that the assembler outputs is up to 255 different-name segments, with segments defined by the segment definition directive. The segments having the same name are counted as one.
- The maximum number of characters recognizable as a segment name is 8.
- Segment names are case sensitive.

ORG

origin

ORG

[Application examples]

```

        NAME          SAMP1
        LOCATION      0H

        DSEG
        ORG           0FE20H          ; (1)
SADR1 : DS          1
SADR2 : DS          1
SADR3 : DS          2

MAIN0  ORG          100H
        MOV          A, SADR1        ; (2)

        CSEG
MAIN1  ORG          1000H            ; (4)
        MOV          A, SADR2
        MOVW         AX, SADR3
        END

```

<Explanation>

- (1) An absolute segment that belongs to a data segment is defined. This absolute segment will be located from the short direct addressing area that starts from address "FE20H".
Because specification of the segment name is omitted, the assembler automatically assigns the name "?A00FE20".
- (2) Because no instruction can be described within an absolute segment that belongs to a data segment, an error occurs.
- (3) This directive declares the start of a code segment.
- (4) This absolute segment is located in an area that starts from address "1000H".

3.3 Symbol Definition Directives

Symbol definition directives assign names to numerical data to be used for describing a source module. These names clarify the meaning of each data value and make the contents of the source module easy to understand.

Symbol definition directives inform the assembler of the value of each name to be used in the source module.

Two directives EQU and SET are available for symbol definition.

EQU

equate

EQU

(1) EQU (equate)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
name	EQU	expression	[:comment]

[Function]

- The EQU directive defines a name that has the value and attributes (symbol attribute and relocation attribute) of the expression specified in the operand field.

[Use]

- Define numerical data to be used in the source module as a name with the EQU directive and describe the name in the operand of an instruction in place of the numerical data.
Numerical data to be frequently used in the source module is recommended to be defined as a name. If you must change a data value in the source module, all you need to do is to change the operand value of the name (see **Application example**).

[Explanation]

- When a name or label is to be described in the operand of the EQU directive, use the name or label that has already been defined in the source module.
No external reference term can be described as the operand of this directive.
- An expression including a term created by a HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS operator that has a relocatable term in its operand cannot be described.
- If an expression with any of the following patterns of operands is described, an error will result.
 - Expression 1 with ADDRESS attribute – expression 2 with ADDRESS attribute
 - Expression 1 with ADDRESS attribute Relational operator Expression 2 with ADDRESS attribute
 - Either of the following conditions <1> and <2> is fulfilled in the above expression (a) or (b).
 - <1> If label 1 in the expression 1 with ADDRESS attribute and label 2 in the expression 2 with ADDRESS attribute belong to the same segment and if a BR directive for which the number of bytes of the object code cannot be determined is described between the two labels
 - <2> If label 1 and label 2 differ in segment and if a BR directive for which the number of bytes of the object code cannot be determined is described between the beginning of the segment and label
 - HIGH absolute expression with ADDRESS attribute
 - LOW absolute expression with ADDRESS attribute
 - HIGHW absolute expression with ADDRESS attribute
 - LOWW absolute expression with ADDRESS attribute
 - DATAPOS absolute expression with ADDRESS attribute
 - BITPOS absolute expression with ADDRESS attribute
 - The <3> below is fulfilled in the expression (d), (e), (f), (g), (h), or (i)
 - <3> If a BR directive for which the number of bytes of the object code cannot be determined instantly is described between the label in the expression with ADDRESS attribute and the beginning of the segment to which the label belongs

EQU

equate

EQU

- If an error exists in the description format of the operand, the assembler will output an error message, but will attempt to store the value of the operand as the value of the name described in the symbol field to the extent that it can analyze.
- A name defined with the EQU directive cannot be redefined within the same source module.
- A name that has defined a bit value with the EQU directive will have an address and bit position as value.
- **Table 3-9 Representation Formats of Operands Indicating Bit Values** shows the bit values that can be described as the operand of the EQU directive and the range in which these bit values can be referenced.

Table 3-9. Representation Formats of Operands Indicating Bit Values

Operand Type	Symbol Value	Reference Range
A.bit ^{Note 1}	1.bit1	Can be referenced within the same module only.
X.bit ^{Note 1}	0.bit1	
PSWL.bit ^{Note 1}	1FEH.bit1	
PSWH.bit ^{Note 1}	1FFH.bit1	
sfr ^{Note 2} .bit ^{Note 1}	00FFxxH ^{Note 3} .bit1	
saddr.bit ^{Note 1}	0nnnnnnH ^{Note 4} .bit1	Can be referenced from another module.
expression.bit ^{Note 1}	0xxxxH ^{Note 3} .bit1	

- Notes**
1. bit1 = 0 to 7
 2. For a detailed description, refer to the user's manual of each device.
 3. "0xFFxxH" denotes the address of an sfr (depending on the LOCATION instruction) and "0xxxxH" denotes the value of an expression.
 4. "0nnnnnnH" denotes the saddr area.

EQU

equate

EQU

[Application example]

```

NAME      SAMP1
LOCATION   0FH

WORK1 EQU  0FFE20H      ; (1)
WORK10 EQU WORK1.0     ; (2)
P02 EQU   P0.2         ; (3)
A4 EQU    A.4          ; (4)
X5 EQU    X.5          ; (5)
PSWL5 EQU PSWL.5       ; (6)
PSWH6 EQU PSWH.6       ; (7)

MOV1 CY, WORK10        ; (8)
MOV1 P0.2, CY          ; (9)
OR1  CY, A4            ; (10)
XOR1 CY, X5            ; (11)
SET1 PSWL5             ; (12)
CLR1 PSWH6             ; (13)

END

```

<Explanation>

- (1) The name "WORK1" has the value "0FFE20H", symbol attribute "NUMBER", and relocation attribute "ABSOLUTE".
- (2) The name "WORK10" is assigned to bit value "WORK1.0", which is in the operand format "saddr.bit". "WORK1", which is described in an operand, is already defined at the value "0FFE20H", in (1) above.
- (3) The name "P02" is assigned to the bit value "P0.2" which is in the operand format "sfr.bit".
- (4) The name "A4" is assigned to the bit value "A.4" which is in the operand format "A.bit".
- (5) The name "X5" is assigned to the bit value "X.5" which is in the operand format "X.bit".
- (6) The name "PSWL5" is assigned to the bit value "PSWL.5" which is in the operand format "PSWL.bit".
- (7) The name "PSWH6" is assigned to the bit value "PSWH.6" which is in the operand format "PSWH.bit".
- (8) This description corresponds to "MOV1 CY, saddr.bit".
- (9) This description corresponds to "MOV1 sfr.bit, CY".
- (10) This description corresponds to "OR1 CY, A.bit".
- (11) This description corresponds to "XOR1 CY, X.bit".
- (12) This description corresponds to "SET1 PSWL.bit".
- (13) This description corresponds to "CLR1 PSWH.bit".

Names in which "sfr.bit", "A.bit", "X.bit", "PSWL.bit", and "PSWH.bit" are defined as in (3) through (7) can be referenced only within the same module.

A name in which "saddr.bit" is defined can also be referenced from another module as an external definition symbol (see **3.5 (2) EXTBIT**).

As a result of assembling the source module in example, the following assemble list is generated.

EQU

equate

EQU

Assemble list						
ALNO	STNO	ADRS	OBJECT	M I	SOURCE STATEMENT	
1	1				NAME SAMP2	
2	2	000000	09C1FF00		LOCATION 0FH	
3	3					
4	4		(000FFE20)	WORK1	EQU 0FFE20H	;(1)
5	5		(0FFE20.0)	WORK10	EQU WORK1.0	;(2)
6	6		(00FF00.2)	P02	EQU P0.2	;(3)
7	7		(000001.4)	A4	EQU A.4	;(4)
8	8		(000000.5)	X5	EQU X.5	;(5)
9	9		(0001FE.5)	PSWL5	EQU PSWL.5	;(6)
10	10		(0001FE.6)	PSWH6	EQU PSWH.6	;(7)
11	11					
12	12	000004	3C080020		MOV1 CY,WORK10	;(8)
13	13	000008	081200		MOV1 P02,CY	;(9)
14	14	00000B	034C		OR1 CY,A4	;(10)
15	15	00000D	0365		XOR1 CY,X5	;(11)
16	16	00000F	0285		SET1 PSWL5	;(12)
17	17	000011	0296		CLR1 PSWH6	;(13)
18	18					
19	19				END	

<Explanation>

On lines (2) through (7) of the assemble list, the bit address values of the bit values defined as names are indicated in the object code field.

SET

set

SET

(2) SET (set)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
name	SET	absolute-expression	[:comment]

[Function]

- The SET directive defines a name that has the value and attributes (symbol attribute and relocation attribute) of the expression specified in the operand field.
- The value and attribute of a name defined with the SET directive can be redefined within the same module. These values and attribute are valid until the same name is redefined.

[Use]

- Define numerical data (a variable) to be used in the source module as a name and describe it in the operand of an instruction in place of the numerical data (a variable).
To change the value of a name in the source module, a different value can be defined for the same name using the SET directive again.

[Explanation]

- An absolute expression must be described in the operand field of the SET directive.
- The SET directive may be described anywhere in a source program. However, a name that has been defined with the SET directive cannot be forward-referenced.
- If an error is detected in the statement in which a name is defined with the SET directive, the assembler outputs an error message but will attempt to store the value of the operand as the value of the name described in the symbol field to the extent that it can analyze.
- A symbol defined with the EQU directive cannot be redefined with the SET directive.
A symbol defined with the SET directive cannot be redefined with the EQU directive.
- A bit symbol cannot be defined.

SET

set

SET

[Application example]

	NAME	SAMP1	
	LOCATION	0FH	
COUNT	SET	10H	; (1)
	CSEG		
	MOV	B, #COUNT	; (2)
LOOP:			
	DEC	B	
	BNZ	\$LOOP	
COUNT	SET	20H	; (3)
	MOV	B, #COUNT	; (4)
	END		

<Explanation>

- (1) The name "COUNT" has the value "10H", the symbol attribute "NUMBER", and relocation attribute "ABSOLUTE". The value and attributes are valid until they are redefined by the SET directive in (3) below.
- (2) The value "10H" of the name "COUNT" is transferred to register B.
- (3) The value of the name "COUNT" is changed to "20H".
- (4) The value "20H" of the name "COUNT" is transferred to register B.

3.4 Memory Initialization and Area Reservation Directives

Memory initialization directives define the constant data to be used in a source program.

The values of the defined constant data are generated as object codes.

Area reservation directives reserve memory areas to be used in a program.

DB

define byte

DB

(1) DB (define byte)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	DB	{(size) initial-value [...]}	[:comment]

[Function]

- The DB directive tells the assembler to initialize a byte area. The number of bytes to be initialized can be specified as “size”.
- The DB directive also tells the assembler to initialize a memory area in byte units with the initial value(s) specified in the operand field.

[Use]

- Use the DB directive when defining an expression or character string used in the program.

[Explanation]

- If a value in the operand field is parenthesized, the assembler assumes that a size is specified. Otherwise, an initial value is assumed.
- The DB directive cannot be described in a bit segment.

With size specification:

- If a size is specified in the operand field, the assembler initializes an area equivalent to the specified number of bytes with the value “00H”.
- An absolute expression must be described as a size. If the size description is illegal, the assembler outputs an error message and will not execute initialization.

With initial value specification:

- The following two parameters can be specified as initial values:
 - <1> Expression
 - The value of an expression must be 8-bit data. Therefore, the value of the operand must be in the range of 0H to 0FFH. If the value exceeds 8 bits, the assembler will use only the lower 8 bits of the value as valid data and output an error message.
 - <2> Character string
 - If a character string is described as the operand, an 8-bit ASCII code will be reserved for each character in the string.
- Two or more initial values may be specified within a statement line of the DB directive.
- As an initial value, an expression that includes a relocatable symbol or external reference symbol may be described.

DB

define byte

DB

[Application example]

```

        NAME          SAMP1
        LOCATION      0FH
        CSEG
WORK1:  DB           (1)                ; (1)
WORK2:  DB           (2)                ; (1)
        CSEG
MASSAG: DB           'ABCDEF'          ; (2)
DATA1:  DB           0AH, 0BH, 0CH     ; (3)
DATA2:  DB           (3+1)             ; (4)
DATA3:  DB           'AB' + 1         ; (5)

        END

```

<Explanation>

- (1) Because the size is specified, the assembler will initialize each byte area with the value "00H".
- (2) A 6-byte area is initialized with character string 'ABCDEF'.
- (3) A 3-byte area is initialized with "0AH, 0BH, 0CH".
- (4) A 4-byte area is initialized with "00H".
- (5) Because the value of expression 'AB' + 1 is 4143H (4142H+1) and exceeds the range of 0 to 0FFH, this description will result in an error.

DW

define word

DW

(2) DW (define word)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	DW	{(size) initial-value [...]}	[:comment]

[Function]

- The DW directive tells the assembler to initialize a word area. The number of words to be initialized can be specified as “size”.
- The DW directive also tells the assembler to initialize a memory area in word units (2 bytes) with the initial value(s) specified in the operand field.

[Use]

- Use the DW directive when defining a 16-bit numeric constant such as an address or data used in the program.

[Explanation]

- If a value in the operand field is parenthesized, the assembler assumes that a size is specified; otherwise an initial value is assumed.
- The DW directive cannot be described in a bit segment.

With size specification:

- If a size is specified in the operand field, the assembler will initialize an area equivalent to the specified number of words with the value “00H”.
- An absolute expression must be described as a size. If the size description is illegal, the assembler outputs an error message and will not execute initialization.

With initial value specification:

- The following two parameters can be specified as initial values:
 - <1> Constant
16 bits or less.
 - <2> Expression
The value of an expression must be stored as a 16-bit data.
No character string can be described as an initial value.
- The upper 2 digits of the specified initial value are stored in the HIGH address and the lower 2 digits of the value in the LOW address.
- Two or more initial values may be specified within a statement line of the DW directive.
- As an initial value, an expression that includes a relocatable symbol or external reference symbol may be described.

DW

define word

DW

[Application example]

```

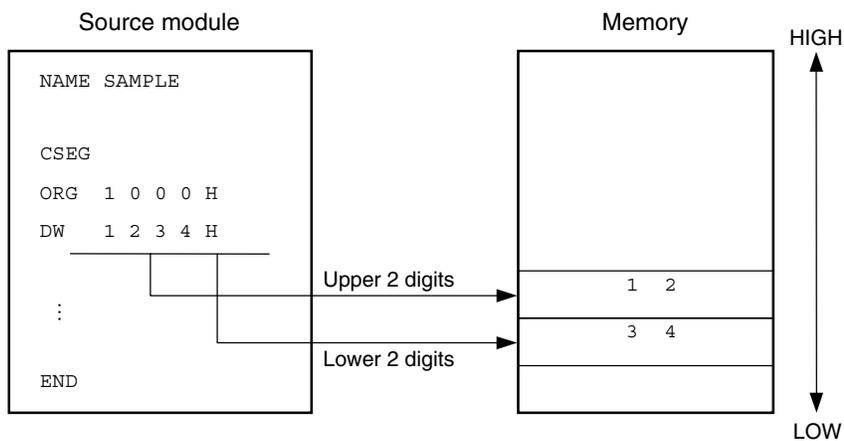
NAME          SAMP1
LOCATION       0FH
CSEG
WORK1: DW      (10)          ; (1)
WORK2: DW      (128)        ; (1)
CSEG
ORG          10H
DW           MAIN           ; (2)
DW           SUB1          ; (2)
CSEG
MAIN:
CSEG
SUB1:
DATA:  DW      1234H,5678H  ; (3)
END

```

<Explanation>

- (1) Because the size is specified, the assembler will initialize each word with the value "00H".
- (2) Vector entry addresses are defined with the DW directives.
- (3) A 2-word area is initialized with value "34127856".

Caution The HIGH address of memory is initialized with the upper 2 digits of the word value. The LOW address of memory is initialized with the lower 2 digits of the word value.

Example:

DG

dg

DG

(3) DG (dg)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	DG	{{(size) initial-value [...]}}	[:comment]

[Function]

- The DG directive tells the assembler to initialize a 3-byte area. The initial value or size can be specified as the operand.
- The DG directive also tells the assembler to initialize a memory area in units of 3 bytes with the initial value(s) specified in the Operand field.

[Use]

- Use the DG directive when defining a 24-bit numeric constant such as an address or a data used in the program.

[Explanation]

- If a value in the operand field is parenthesized, the assembler assumes that a size is specified. Otherwise, an initial value is assumed.
- The DG directive cannot be described in a bit segment.

With size specification:

- If a size is specified in the operand field, the assembler will initialize an area equivalent to the specified number x 3 bytes with the value "00H".
- An absolute expression must be described as a size. If the size description is illegal, the assembler will output an error message and will not execute initialization.

With initial value specification:

- The following two parameters can be specified as initial values:
 - 1) Constant
24 bits or less.
 - 2) Expression
The value of an expression must be stored as a 24-bit data.
No character string can be described as an initial value.
- The most significant byte of the initial value is stored in the HIGH WORD address^{Note} and the least significant byte of the value in the LOW address. The highest byte of the lowest 2 bytes is reserved in the HIGH address.
- Two or more initial values may be specified within one statement line of the DG directive.
- As an initial value, an expression which includes a relocatable symbol or external reference symbol is described.

Note HIGH WORD is a 1-byte address.

[Application example]

```

NAME      SAMP1
LOCATION   0FH
DATA1: DG      123456H,567890H      ; (1)
DATA2: DG      (10)                ; (2)

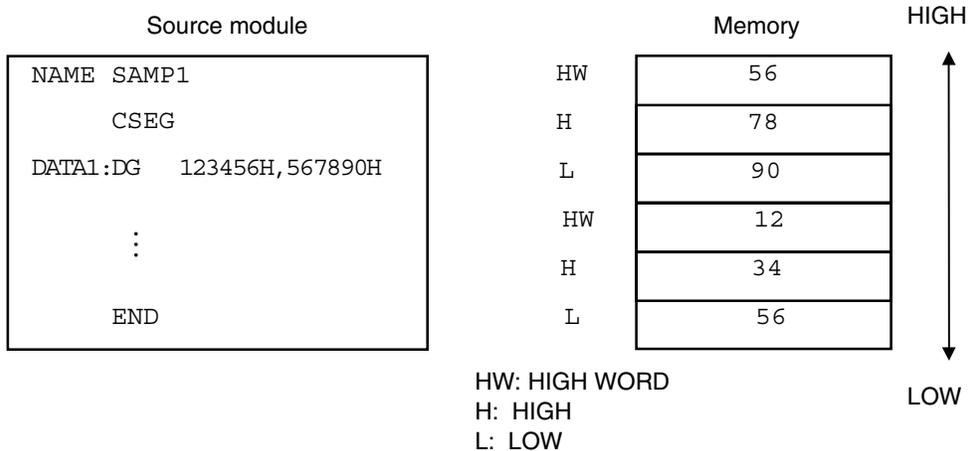
END
    
```

<Explanation>

- (1) A 3-byte area is initialized with value "563412907856".
- (2) A 30-byte area (10 x 3 bytes) is initialized with "00H".

Caution The HIGH WORD address of memory is initialized with the most significant byte of 3-byte value. The LOW address and the HIGH address of memory are initialized with the least significant byte and the highest byte of the 2-byte value, respectively.

Example:



DS

define storage

DS

(4) DS (define storage)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	DS	absolute-expression	[:comment]

[Function]

- The DS directive tells the assembler to reserve a memory area for the number of bytes specified in the operand field.

[Use]

- The DS directive is mainly used to reserve a memory (RAM) area to be used in the program. If a label is specified, the value of the first address of the reserved memory area is assigned to the label. In the source module, this label is used for description to manipulate the memory.

[Explanation]

- The contents of an area to be reserved with this DS directive are unknown (indefinite).
- The specified absolute expression will be evaluated with unsigned 16 bits.
- When the operand value is “0”, no area can be reserved.
- The DS directive cannot be described within a bit segment.
- The symbol (label) defined with the DS directive can be referenced only in the backward direction.
- Only the following parameters extended from an absolute expression can be described in the operand field.
 - <1> A constant
 - <2> An expression with constants in which an operation is to be performed (constant expression)
 - <3> EQU symbol or SET symbol defined with a constant or constant expression
 - <4> Expression 1 with ADDRESS attribute – expression 2 with ADDRESS attribute
 If both label 1 in “expression 1 with ADDRESS attribute” and label 2 in “expression 2 with ADDRESS attribute” are relocatable, both labels must be defined in the same segment.
 However, an error will result in either of the following two cases:
 - (a) If label 1 and label 2 belong to the same segment and if a BR directive for which the number of bytes of the object code cannot be determined is described between the two labels
 - (b) If label 1 and label 2 differ in segment and if a BR directive for which the number of bytes of the object code cannot be determined is described between either label and the beginning of the segment to which the label belongs
 - <5> Any of the expressions <1> through <4> above on which an operation is to be performed.
- The following parameters cannot be described in the operand field.
 - <1> External reference symbol
 - <2> Symbol that has defined “expression 1 with ADDRESS attribute – expression 2 with ADDRESS attribute” with the EQU directive
 - <3> Location counter (\$) is described in either expression 1 or expression 2 in the form of “expression 1 with ADDRESS attribute – expression 2 with ADDRESS attribute”
 - <4> Symbol that defines with the EQU directive an expression with the ADDRESS attribute on which the HIGH/LOW/DATAPOS/BITPOS operator is to be operated

DS

define storage

DS

[Application example]

```
        NAME      SAMPLE
        DSEG
TABLE1: DS      10          ; (1)
WORK1: DS       1          ; (2)
WORK2: DS       2          ; (3)
        CSEG
        MOVW     HL, #TABLE1
        MOV      A, !WORK1
        MOVW     BC, #WORK2
        END
```

<Explanation>

- (1) A 10-byte working area is reserved, but the contents of the area are unknown (indefinite). Label "TABLE1" is allocated to the start of the address.
- (2) A 1-byte working area is reserved.
- (3) A 2-byte working area is reserved.

DBIT

define bit

DBIT

(5) DBIT (define bit)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[name]	DBIT	None	[:comment]

[Function]

- The DBIT directive tells the assembler to reserve a 1-bit memory area within a bit segment.

[Use]

- Use the DBIT directive to reserve a bit area within a bit segment.

[Explanation]

- The DBIT directive is described only in a bit segment.
- The contents of a 1-bit area reserved with the DBIT directive are unknown (indefinite).
- If a name is specified in the symbol field, the name has an address and a bit position as its value.

[Application Example]

```

NAME      SAMPLE
BSEG
BIT1      DBIT                ; (1)
BIT2      DBIT                ; (1)
BIT3      DBIT                ; (1)

CSEG
MOV1      CY, BIT1            ; (2)

OR1       CY, BIT2            ; (3)

END

```

<Explanation>

- (1) By these three DBIT directives, the assembler will reserve three 1-bit areas and define names (BIT1, BIT2, and BIT3) each having an address and a bit position as its value.
- (2) This description corresponds to "MOV1 CY,saddr.bit" and describes the name "BIT1" of the bit area reserved in (1) above as operand "saddr.bit".
- (3) This description corresponds to "OR1 CY, saddr.bit" and describes name "BIT2" as "saddr.bit".

3.5 Linkage Directives

Linkage directives clarify the relativity to reference a symbol defined in the other modules.

Consider a case where a program is created by being divided into two modules: module 1 and module 2. If a symbol defined in module 2 is to be referenced in module 1, the symbol cannot be used without declaration in each module. For this reason, some sort of signal or indication such as “I want to use the symbol” or “You may use the symbol” is required to be issued between the two modules.

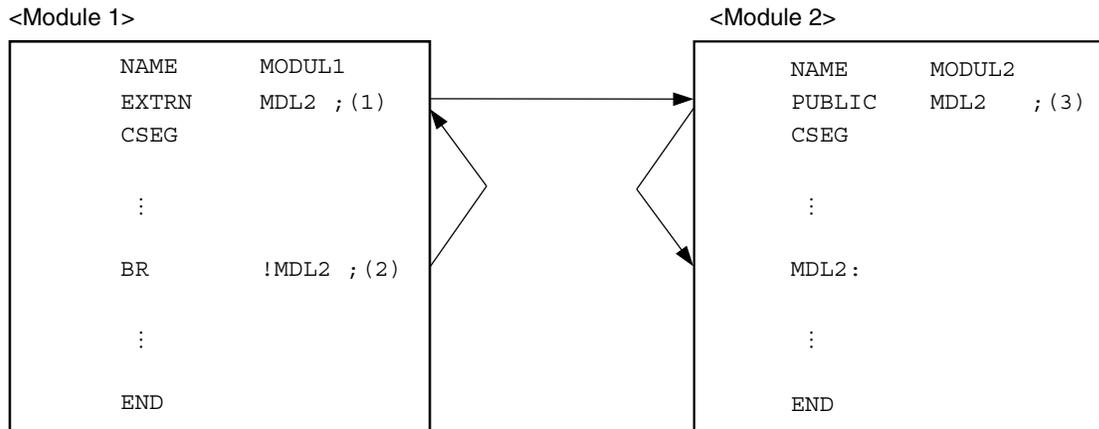
In module 1, the external reference of a symbol to indicate “I want to reference a symbol defined in another module” must be declared. In module 2, the external definition of a symbol to indicate “You may reference the defined symbol in another module” must be declared.

The symbol can be referenced for the first time when both the external reference and the external definition are effectively declared.

Linkage directives function to establish this interrelationship and are available in the following two types.

- To declare external definition of a symbol: PUBLIC directive
- To declare external reference of a symbol: EXTRN and EXTBIT directives

Figure 3-6. Relationship of Symbols Between Two Modules



In module 1 in Figure 3-6, the symbol “MDL2” defined in module 2 is referenced in (2). Therefore, the symbol is declared as an external reference with the EXTRN directive in (1).

In module 2, the symbol “MDL2” to be referenced from module 1 is declared as an external definition with the PUBLIC directive in (3).

The linker checks whether or not the external reference of the symbol corresponds to the external definition of the symbol.

EXTRN

external

EXTRN

(1) EXTRN (external)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	EXTRN	{symbol-name [...] SADDR2 (symbol-name [...]) BASE (symbol-name [...])}	[:comment]

[Function]

- The EXTRN directive declares to the linker that a symbol (other than a bit symbol) in another module is to be referenced in this module.

[Use]

- When referencing a symbol defined in another module, the EXTRN directive must be used to declare the symbol as an external reference.
- There are following differences depending on the description format of the operand.

SADDR2 (symbol-name [...])	The symbol can be referenced as saddr2 area.
BASE (symbol-name [...])	The symbol can be referenced as that of an area within 64 KB (0H to 0FFFFH).
No relocation attribute	The symbol can be referenced after the segment has been relocated by the link to match the area of the symbol declared with PUBLIC.

[Explanation]

- The EXTRN directive may be described anywhere in a source program (see **2.1 Basic Configuration of Source Program**).
- Up to 20 symbols can be specified in the operand field by delimiting each symbol name with a comma (,).
- When referencing a symbol having a bit value, the symbol must be declared as an external reference with the EXTBIT directive.
- The symbol declared with the EXTRN directive must be declared in another module with a PUBLIC directive.
- No macro name can be described as the operand of EXTRN directive (see **CHAPTER 5 MACROS** for the macro name).
- The EXTRN directive enables only one EXTRN declaration for a symbol in an entire module. For the second and subsequent EXTRN declarations for the symbol, the linker will output a warning message.
- A symbol that has been declared cannot be described as the operand of the EXTRN directive. Conversely, a symbol that has been declared as EXTRN cannot be re-defined or declared with any other directive.
- A symbol defined by the EXTRN directive can be used to reference saddr area.

EXTRN

external

EXTRN

[Application example]

<Module 1>

```

        NAME      SAMP1
        LOCATION  0FH
        EXTRN    SYM1 , SYM2 , SADDR2 (SYM3) , BASE (SYM4)      ; (1)
        CSEG
S1:    DW      SYM1      ; (2)
        MOV     A, SYM2      ; (3)
        MOV     A, SYM3      ; (4)
        BR     !SYM4      ; (5)
        END

```

<Module 2>

```

        NAME      SAMP2
        PUBLIC   SYM1 , SYM2 , SYM3 , SYM4      ; (5)
        CSEG
SYM1   EQU     0FFH      ; (6)
DATA1  DSEG    SADDR
SYM2:  DB     012H      ; (7)
DATA2  DSEG    SADDR2
SYM3:  DB     034H      ; (8)
C1     CSEG    BASE
SYM4:  MOV     A, #20H   ; (9)
        END

```

<Explanation>

- (1) This EXTRN directive declares the symbols "SYM1", "SYM2", "SYM3", and "SYM4" to be referenced in (2) and (3) as external references. Two or more symbols may be described in the operand field.
- (2) This DW instruction references the symbol "SYM1".
- (3) This MOV instruction references the symbol "SYM2" and outputs a code that references saddr2 area.
- (4) This MOV instruction references the symbol "SYM3" and outputs a code that references an area within 64 KB (0H to 0FFFFH).
- (5) The symbols "SYM1", "SYM2", "SYM3", and "SYM4" are declared as external definitions.
- (6) The symbol "SYM1" is defined.
- (7) The symbol "SYM2" is defined.
- (8) The symbol "SYM3" is defined.
- (9) The symbol "SYM4" is defined.

EXTBIT

external bit

EXTBIT

(2) EXTBIT (external bit)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	EXTBIT	bit-symbol-name [...] SADDR2 (symbol-name [...]) SADDRA (symbol-name [...])	[:comment]

[Function]

- The EXTBIT directive declares to the linker that a bit symbol that has a value of saddr.bit in another module is to be referenced in this module.

[Use]

- When referencing a symbol that has a bit value and has been defined in another module, the EXTBIT directive must be used to declare the symbol as an external reference.

[Explanation]

- The EXTBIT directive may be described anywhere in a source program.
- Up to 20 symbols can be specified in the operand field by delimiting each symbol with a comma (,).
- A symbol declared with the EXTBIT directive must be declared with a PUBLIC directive in another module.
- The EXTBIT directive enables only one EXTBIT declaration for a symbol in an entire module. For the second and subsequent EXTBIT declarations for the symbol, the linker will output a warning message.

[Application example]

<Module 1>

```

NAME      SAMP1
EXTBIT    FLAG1 , SADDR2 ( FLAG2 , FLAG3 ) , FLAG4      ; (1)
CSEG
MOV1      FLAG1 , CY                                     ; (2)
AND1      CY , FLAG2                                    ; (3)
SET1      FLAG3                                         ; (4)
NOT1      FLAG4                                         ; (5)
END

```

<Module 2>

```

NAME      SAMP2
PUBLIC    FLAG1 , FLAG2 , FLAG3 , FLAG4                ; (6)
B1        BSEG      SADDR
FLAG1     DBIT                                           ; (7)
FLAG4     DBIT                                           ; (8)
B2        BSEG      SADDR2
FLAG2     DBIT                                           ; (9)
FLAG3     DBIT                                           ; (10)
CSEG
END

```

<Explanation>

- (1) This EXTBIT directive declares the symbols "FLAG1", "FLAG2", "FLAG3", and "FLAG4" to be referenced as external references. Two or more symbols may be described in the operand field.
- (2) This MOV1 instruction references the symbol "FLAG1". This description corresponds to "MOV1 saddr1.bit, CY".
- (3) This AND1 instruction references the symbol "FLAG2". This description corresponds to "AND1 CY, saddr2.bit".
- (4) This SET1 instruction references the symbol "FLAG3". This description corresponds to "SET1 saddr2.bit".
- (5) This NOT1 instruction references the symbol "FLAG4". This description corresponds to "NOT1 saddr1.bit".
- (6) This PUBLIC directive defines the symbols "FLAG1", "FLAG2", "FLAG3" and "FLAG4".
- (7) This DBIT directive defines the symbol "FLAG1" as a bit symbol of SADDR1 area.
- (8) This DBIT directive defines the symbol "FLAG4" as a bit symbol of SADDR1 area.
- (9) This DBIT directive defines the symbol "FLAG2" as a bit symbol of SADDR2 area.
- (10) This DBIT directive defines the symbol "FLAG3" as a bit symbol of SADDR2 area.

PUBLIC

public

PUBLIC

(3) PUBLIC (public)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	PUBLIC	symbol-name [...]	[:comment]

[Function]

- The PUBLIC directive declares to the linker that the symbol described in the operand field is a symbol to be referenced from another module.

[Use]

- When defining a symbol (including bit symbol) to be referenced from another module, the PUBLIC directive must be used to declare the symbol as an external definition.

[Explanation]

- The PUBLIC directive may be described anywhere in a source program.
- Up to 20 symbols can be specified in the operand field by delimiting each symbol name with a comma (,).
- Symbol(s) to be described in the operand field must be defined within the same module.
- The PUBLIC directive enables only one PUBLIC declaration for a symbol in an entire module. The second and subsequent PUBLIC declarations for the symbol will be ignored by the linker.
- The following symbols cannot be used as the operand of the PUBLIC directive.
 - Name defined with the SET directive
 - Symbol defined with the EXTRN or EXTBIT directive within the same module
 - Segment name
 - Module name
 - Macro name
 - Symbol not defined within the module
 - Symbol defining an operand with a bit attribute with the EQU directive
 - Symbol defining an sfr with the EQU directive (however, the place where sfr area and saddr area are overlapped is excluded)

PUBLIC

public

PUBLIC

[Application example]

Example of program consisting of three modules

<Module 1>

```

NAME      SAMP1
PUBLIC   A1, A2                ; (1)
EXTRN   B1
EXTBIT   C1

A1      EQU      10H
A2      EQU      0FFE20H.1

CSEG

BR       B1
XOR1    CY, C1
END

```

<Module 2>

```

NAME      SAMP2
PUBLIC   B1                    ; (2)
EXTRN   A1
CSEG

B1:
MOV     C, #LOW(A1)
END

```

<Module 3>

```

NAME      SAMP3
PUBLIC   C1                    ; (3)
EXTBIT   A2
C1      EQU      0FFE21H.0
CSEG

MOV1    CY, A2
END

```

<Explanation>

- (1) This PUBLIC directive declares that the symbols "A1" and "A2" are to be referenced from other modules.
- (2) This PUBLIC directive declares that the symbol "B1" is to be referenced from another module.
- (3) This PUBLIC directive declares that the symbol "C1" is to be referenced from another module.

3.6 Object Module Name Declaration Directive

The object module name declaration directive gives a module name to an object module to be created by the RA78K4 assembler.

NAME

name

NAME

(1) NAME (name)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	NAME	object-module-name	[:comment]

[Function]

- The NAME directive assigns the object module name described in the operand field to an object module to be output by the assembler.

[Use]

- A module name is required for each object module in symbolic debugging with a debugger.

[Explanation]

- The NAME directive may be described anywhere in a source program.
- For the conventions of module name description, see the conventions on symbol description in **2.2.3 Fields that make up a statement**.
- Characters that can be specified as a module name are those characters permitted by the operating system of the assembler software other than “(”, “(28H)”, “)” or “(29H)”.
- No module name can be described as the operand of any directive other than NAME or of any instruction.
- If the NAME directive is omitted, the assembler will assume the primary name (first 8 characters) of the input source module file as the module name. In the Windows version, the primary name is converted to capital letters for retrieval. If two or more module names are specified, the assembler will output a warning message and ignore the second and subsequent module name declarations.
- A module name to be described in the operand field must not exceed eight characters.
- Symbol names are case sensitive.

[Application example]

```

NAME    SAMPLE                ; (1)
DSEG
BIT1:  DBIT

CSEG
MOV    A, B

END

```

<Explanation>

- (1) This NAME directive declares “SAMPLE” as a module name.

3.7 Automatic Branch Instruction Selection Directive

Unconditional branch instructions directly describe a branch destination address as their operand. Four such instructions, “BR !addr16”, “BR \$addr20”, “BR \$!addr20”, and “BR !!addr20”, are available. Also, three such instructions, “CALL !!addr20”, “CALL \$!addr20”, and “CALL !addr16”, are available for the CALL instruction. These instructions select and use the most appropriate operand according to the address range of the branch destination. Since the number of bytes is different for each directive, in order to create a program with high memory utilization efficiency, it is necessary to use the instruction with the smallest number of bytes. However, it is quite troublesome to take this address range into account when describing the branch instruction.

For this reason, there was a need for a directive that directs the assembler to automatically select the two-byte or three-byte branch instruction according to the address range of the branch destination. This is called automatic branch instruction selection directive.

BR

branch

BR

(1) BR (branch)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	BR	expression	[:comment]

[Function]

- The BR directive tells the assembler to automatically select a 2- or 3-byte BR branch instruction according to the value range of the expression specified in the operand field and to generate the object code applicable to the selected instruction.

[Use]

- The BR directive judges the address range of the branch destination and automatically selects the smallest possible branch instruction from among the four branch instructions below. Use the BR directive if it is unclear whether or not a 2-byte branch instruction can describe the address range of the branch destination.

"BR \$addr20" (2 bytes) ... This instruction can be used within a range of between -80H and +7FH from the next address of the BR directive.

"BR \$addr16" (3 bytes) ... This instruction can be used within 64 KB.

"BR !\$addr20" (3 bytes) ... This instruction calculates the displacement between source and destination addresses. The displacement must be between -8000H and +7FFFH.

"BR !!addr20" (4 bytes) ... Use this instruction in cases other than the above.

If the operand (branch destination) is allocated outside the BASE area within a relocatable segment that is different from the directive, the BR branch instruction is replaced with a 4-byte instruction and output.

When the directive and operand (branch destination) are different segments, allocated outside the BASE area, and are separate types^{Note}, the BR branch instruction is replaced with a 4-byte instruction even if the operand is allocated within an absolute segment.

If the directive and operand (branch destination) are in separate segments within the BASE area, the BR branch instruction is replaced with a 3-byte instruction (BR !addr16).

Note "Separate type" indicates a separate relocatable segment if the BR directive is within an absolute segment, and an absolute segment if the BR directive is a relocatable segment.

- If it is definite that you can describe a 2-byte to 4-byte instruction, describe the applicable instruction. This shortens the assembly time in comparison with describing the BR directive.

BR

branch

BR

[Explanation]

- The BR directive can only be used within a code segment.
 - The direct jump destination is described as the operand of the BR directive. “\$” indicating the current location counter at the beginning of an expression cannot be described.
 - For optimization, the following conditions must be satisfied.
 - <1> No more than 1 label or forward-reference symbol in the expression.
 - <2> Do not describe an EQU symbol with the ADDRESS attribute.
 - <3> Do not describe an EQU defined symbol for “expression 1 with ADDRESS attribute – expression 2 with ADDRESS attribute”.
 - <4> Do not describe an expression with ADDRESS attribute on which the HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS operator has been operated.
- If these conditions are not met, the 4-byte BR instruction will be selected.

[Application example]

ADDRESS		NAME	SAMPLE	
	C1	CSEG	AT	50H
000050H	BR	L1		; (1)
000052H	BR	L2		; (2)
000055H	BR	L3		; (3)
000058H	BR	L4		; (4)
00007DH	L1 :			
007FFFH	L2 :			
00FFFFH	L3 :			
010000H	L4 :			
				END

<Explanation>

- (1) This BR directive generates a 2-byte branch instruction (BR \$addr20) because the displacement between this line and the branch destination is within the range of -80H and +7FH.
- (2) This BR directive will be replaced with a 3-byte branch instruction (BR \$!addr20) because the displacement between this line and the branch destination is within the range of -8000H and +7FFFH.
- (3) This BR directive will be replaced with a 3-byte branching instruction (BR !addr16) because the branch destination is within 64 KB.
- (4) This BR directive will be replaced with a 4-byte branch instruction (BR !!addr20).

CALL

call

CALL

(2) CALL (call)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	CALL	expression	[:comment]

[Function]

- The CALL directive tells the assembler to automatically select a 3- or 4-byte CALL instruction according to the value range of the expression specified in the operand field and to generate the object code applicable to the selected instruction.

[Use]

- The CALL directive judges the address range of the branch destination and automatically selects the smallest possible branch instruction from among the three branch instructions below. Use the CALL directive if it is unclear whether or not a 3-byte branch instruction can describe the address range of the branch destination.

"CALL !addr16" (3 bytes) ... This instruction can be used within 64 KB.

"CALL \$!addr20" (3 bytes) ... This instruction calculates the displacement between source and destination addresses. The displacement must be between -8000H and +7FFFH.

"CALL !!addr20" (4 bytes) ... Use this instruction in cases other than the above.

If the operand (branch destination) is allocated within a relocatable segment different from the directive outside the BASE area, the CALL instruction is replaced with a 4-byte instruction and output.

When the directive and operand (branch destination) are not in a single segment, allocated outside the BASE area, and are separate types^{Note}, the CALL instruction is replaced with a 4-byte instruction even if the operand is allocated within an absolute segment.

If the directive and operand (branch destination) are in separate segments within the BASE area, the CALL instruction is replaced with a 3-byte instruction (CALL !addr16).

Note "Separate type" indicates a separate relocatable segment if the CALL directive is within an absolute segment, and an absolute segment if the CALL directive is a relocatable segment.

CALL

call

CALL

[Explanation]

- The CALL directive can only be used within a code segment.
 - The direct call destination is described as the operand of the CALL directive.
 - For optimization, the following conditions must be satisfied.
 - 1) No more than 1 label or forward-reference symbol in the expression.
 - 2) Do not describe an EQU symbol with the ADDRESS attribute.
 - 3) Do not describe an EQU defined symbol for "expression 1 with ADDRESS attribute - expression 2 with ADDRESS attribute".
 - 4) Do not describe an expression with ADDRESS attribute on which the HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS operator has been operated.
- If these conditions are not met, a 4-byte instruction is selected.

[Application example]

ADDRESS			NAME	SAMPLE
	C1	CSEG	AT	50H
000050H		CALL	L1	; (1)
000053H		CALL	L2	; (2)
000056H		CALL	L3	; (3)
008052H	L1 :			
00FFFFH	L2 :			
010000H	L3 :			
			END	

<Explanation>

- (1) This CALL directive will be replaced with a 3-byte branch instruction (CALL \$!addr20) because the displacement between this line and the branch destination is within the range of -8000H and +7FFFH.
- (2) This CALL directive will be replaced with a 3-byte branching instruction (CALL !addr16) because the branching destination is within 64 KB.
- (3) This CALL directive will be substituted with a 4-byte branching instruction (CALL !!addr20).

3.8 General-Purpose Register Selection Directive

With the general-purpose registers of the 78K/IV, the correspondence of their function names to their absolute names is different depending on the value of the Register Set Select (RSS) flag in the PSW (see **Table 3-10**, below).

This means that when you describe the function name of a register in a program in place of its absolute name, the register to be actually accessed differs depending on the value of the RSS flag and that the object code to be generated also differs depending on the value of the RSS flag.

The general-purpose register selection directive informs the assembler of the value set in the RSS flag to generate the object code corresponding to the value of the RSS flag.

Table 3-10. Absolute Names and Function Names of General-Purpose Registers

(a) 8-bit registers

Absolute Name	Function Name	
	RSS = 0	RSS = 1 ^{Note}
R0	X	
R1	A	
R2	C	
R3	B	
R4		X
R5		A
R6		C
R7		B
R8		
R9		
R10		
R11		
R12	E	E
R13	D	D
R14	L	L
R15	H	H

(b) 16-bit registers

Absolute Name	Function name	
	RSS = 0	RSS = 1 ^{Note}
RP0	AX	
RP1	BC	
RP2		AX
RP3		BC
RP4	VP	VP
RP5	UP	UP
RP6	DE	DE
RP7	HL	HL

(c) 24-bit registers

Absolute Name	Function name
RG4	VVP
RG5	UUP
RG6	TDE
RG7	WHL

Note RSS should only be set to 1 when a 78K/III Series program is used.

- Remarks**
1. A blank column in the table indicates that, by describing an absolute name, the register corresponding to the absolute name can be accessed.
 2. R8 to R11 have no function name.

RSS

register set select

RSS

(1) RSS (register set select)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	RSS	absolute-value-with-evaluated-value-of-0-or-1	[:comment]

[Function]

- The RSS directive tells the assembler to generate object codes by replacing the general-purpose registers of the function names described in the source program with those of the corresponding absolute names, based on the value of the Register Set Select (RSS) flag specified in the operand field.

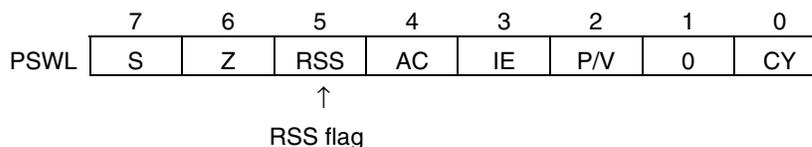
See **Table 3-10 Absolute Names and Function Names of General Registers**, for correspondence of the function names of the general-purpose registers to their absolute names.

[Use]

- When addressing is to be performed by using the function name of a general-purpose register instead of its absolute name to make the best use of its inherent function, use the RSS directive.
- When describing a general-purpose register with its function name, the value then set in the RSS flag must be declared with the RSS directive.

[Explanation]

- The Register Set Select (RSS) flag is bit 5 of the PSWL register.



- The RSS directive informs the assembler of the value (0, 1) of the RSS flag. Based on the value of the operand of the RSS directive, the assembler generates object codes by substituting the general registers of the function names with those of the corresponding absolute names.
- When setting, resetting, or switching the value of the RSS flag with an instruction, the RSS directive must be described immediately before or after the instruction to inform the assembler of the value of the RSS flag. Even after the RSS flag is set or reset by the instruction, the expected object code is not generated unless the RSS directive is described.
- The RSS directive is valid until the next RSS directive, segment definition directive (CSEG, DSEG, BSEG, or ORG), or END directive appears in the source program. Therefore, the RSS directive must be described for each segment.
- The RSS directive can be described only within a code segment.
- If an RSS directive appears while no segment is being created, then the assembler will create a relocatable code segment as a default segment. The default segment name of the created segment is ?CSEG and its default relocation attribute is UNIT.
- The default value of the RSS directive is 0 (RSS = 0).

[Application example]

```

        NAME      SAMPLE
        LOCATION  0FH
        RSS      1          ; (1)
        MOV      B, A      ; (2)
SEG1    CSEG
SUB1:   MOV      B, A      ; (3)
        MOV      A, C      ; (4)
        RET
SEG2    CSEG
SUB2:   RSS      1          ; (5)
        SET1     PSWL.5    ; (6)
        MOV      B, A      ; (7)
        RET
SUB3:   RSS      0          ; (8)
        SWRS     ; (9)
        MOV      B, A      ; (10)
        RET
SUB4:   MOV      B, A      ; (11)
        RET
SEG4    DSEG
VAR:    DW      0
SEG3    CSEG
SUB5:   MOV      B, A      ; (12)
        RET
        END

```

<Explanation>

- (1) A segment is generated.
- (2) This description corresponds to "MOV R7, R5".
- (3) The RSS default value in the assembler is "0". Because there is no description for the RSS directive, this description corresponds to "MOV R3, R1".
- (4) This description corresponds to "MOV R1, R2".
- (5) The RSS directive must be described immediately before (or after) the instruction which sets the RSS flag in (6).
- (7) This description corresponds to "MOV R7, R5".
- (8) The RSS directive must be described immediately before (or after) the instruction which resets the RSS flag in (9).
- (10) This description corresponds to "MOV R3, R1".
- (11) This description corresponds to "MOV R3, R1".
- (12) This description corresponds to "MOV R3, R1".

RSS

register set select

RSS

See the following assemble list for the object codes to be generated.

Assemble list

ALNO	STNO	ADRS	OBJECT	M I	SOURCE STATEMENT
1	1				NAME SAMPLE
2	2	000000	09C1FF00		LOCATION 0FH
3	3				RSS 1 ; (1)
4	4	000004	2475		MOV B,A ; (2)
5	5	-----		SEG1	CSEG
6	6	000000	2431	SUB1:	MOV B,A ; (3)
7	7	000002	D2		MOV A,C ; (4)
8	8	000003	56		RET
9	9				
10	10	-----		SEG2	CSEG
11	11	000000		SUB2:	RSS 1 ; (5)
12	12	000000	0285		SET1 PSWL.5 ; (6)
13	13	000002	2475		MOV B,A ; (7)
14	14	000004	56		RET
15	15	000005		SUB3:	RSS 0 ; (8)
16	16	000005	05FC		SWRS ; (9)
17	17	000007	2431		MOV B,A ; (10)
18	18	000009	56		RET
19	19	00000A	2431	SUB4:	MOV B,A ; (11)
20	20	00000C	56		RET
21	21				
22	22	-----		SEG4	DSEG
23	23	000000	0000	VAR:	DW 0
24	24				
25	25	-----		SEG3	CSEG
26	26	000000	2431	SUB5:	MOV B,A ; (12)
27	27	000002	56		RET
28	28				END

3.9 Macro Directives

When describing a source program, it is not only troublesome to describe a series of frequently used instruction groups over and over again, but this may also cause an increase in the number of description or coding errors.

By using the macro function with macro directives, the need to repeatedly describe the same group of instructions can be eliminated, thereby increasing coding efficiency of the program. The basic function of a macro is the replacement of a series of statements with a name.

Macro directives include MACRO, LOCAL, REPT, IRP, EXITM, and ENDM.

In this section, each of these macro directives is detailed. For details of the macro function, see **CHAPTER 5 MACROS**.

MACRO

macro

MACRO

(1) MACRO (macro)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
macro-name	MACRO	[formal-parameter [...]]	[:comment]
	⋮		
	Macro body		
	⋮		
	ENDM		[:comment]

[Function]

- The MACRO directive executes a macro definition by assigning the macro name specified in the symbol field to a series of statements (called a macro body) described between this directive and the ENDM directive.

[Use]

- Define a series of frequently used statements in the source program with a macro name. After this definition the macro body corresponding to the macro name is expanded by only describing the defined macro name (for macro reference).

[Explanation]

- The MACRO directive must be paired with the ENDM directive.
- For the macro name to be described in the symbol field, see the conventions of symbol description in **2.2.3 Fields that make up a statement**.
- To reference a macro, describe the defined macro name in the mnemonic field (see **Application example**).
- For the formal parameter(s) to be described in the operand field, the same rules as the conventions of symbol description will apply.
- Up to 16 formal parameters can be described per macro directive.
- Formal parameters are valid only within the macro body.
- An error will result if a reserved word is described as a formal parameter. However, if a user-defined symbol is described, its recognition as a formal parameter will take precedence.
- The number of formal parameters must be the same as the number of actual parameters.
- A name or label defined within the macro body if declared with the LOCAL directive becomes valid with respect to one-time macro expansion.
- Nesting of macros (i.e., referencing other macros within the macro body) is allowed up to eight levels including the REPT and IRP directives.
- The number of macros that can be defined within a single source module is not specifically limited. In other words, macros may be defined as long as there is memory space available.
- Formal parameter definition lines, reference lines, and symbol names are not output to a cross-reference list.
- Two or more segments must not be defined in a macro body. If defined, an error will be output.

MACRO

macro

MACRO

[Application example]

```
NAME      SAMPLE
ADMAC    MACRO  PARA1, PARA2          ; (1)
          MOV   A, #PARA1
          ADD   A, #PARA2
          ENDM  ; (2)
ADMAC    10H, 20H                    ; (3)
END
```

<Explanation>

- (1) A macro is defined by specifying macro name "ADMAC" and two formal parameters "PARA1" and "PARA2".
- (2) This directive indicates the end of the macro definition.
- (3) Macro "ADMAC" is referenced.

LOCAL

local

LOCAL

(2) LOCAL (local)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
None	LOCAL	symbol-name [...]	[:comment]

[Function]

- The LOCAL directive declares that the symbol name specified in the operand field is a local symbol that is valid only within the macro body.

[Use]

- If a macro that defines a symbol within the macro body is referenced more than once, the assembler will output a double definition error for the symbol. By using the LOCAL directive, you can reference (or call) a macro that defines symbol(s) within the macro body more than once.

[Explanation]

- For the conventions on symbol names to be described in the operand field, see the conventions on symbol description in **2.2.3 Fields that make up a statement**.
- A symbol declared as LOCAL will be replaced with the symbol “??RAn” (where n = 0000 to FFFF) at each macro expansion. The symbol “??RAn” after the macro replacement will be handled in the same way as a global symbol and will be stored in the symbol table, and can thus be referenced under the symbol name “??RAn”.
- If a symbol is described within a macro body and the macro is referenced more than once, it means that the symbol would be defined more than once in the source module. For this reason, it is necessary to declare that the symbol is a local symbol that is valid only within the macro body.
- The LOCAL directive can be used only within a macro definition.
- The LOCAL directive must be described before using the symbol specified in the operand field (in other words, the LOCAL directive must be described at the beginning of the macro body).
- Symbol names to be defined with the LOCAL directive within a source module must be all different (in other words, the same name cannot be used for local symbols to be used in each macro).
- The number of local symbols that can be specified in the operand field is not limited as long as they are all within a line. However, the number of symbols within a macro body is limited to 64. If 65 or more local symbols are declared, the assembler will output an error message and store the macro definition as an empty macro body. Nothing will be expanded even if the macro is called.
- Macros defined with the LOCAL directive cannot be nested.
- Symbols defined with the LOCAL directive cannot be called (referenced) from outside the macro.
- No reserved word can be described as a symbol name in the operand field. However, if a symbol same as the user-defined symbol is described, its recognition as a local symbol will take precedence.
- A symbol declared as the operand of the LOCAL directive will not be output to a cross-reference list and symbol table list.
- The statement line of the LOCAL directive will not be output at the time of the macro expansion.
- If a LOCAL declaration is made within a macro definition for which a symbol has the same name as a formal parameter of that macro definition, an error will be output.

LOCAL

local

LOCAL

[Application example]

<Source Program>

	NAME	SAMPLE	
MAC1	MACRO		
	LOCAL	LLAB	; (1)
LLAB:			
	BR	\$LLAB	; (2)
	ENDM		
REF1:	MAC1		; (3)
	BR	!LLAB	; (4)
REF2:	MAC1		; (5)
	END		

Macro definition

← This description is erroneous.

<Explanation>

- (1) This LOCAL directive defines the symbol name "LLAB" as a local symbol.
- (2) This BR instruction references the local symbol "LLAB" within the macro MAC1.
- (3) This macro reference calls the macro MAC1.
- (4) Because the local symbol "LLAB" is referenced outside the definition of the macro MAC1, this description results in an error.
- (5) This macro reference calls the macro MAC1.

LOCAL

local

LOCAL

The assemble list of the above application example is shown below.

Assemble list

ALNO	STNO	ADRS	OBJECT	M	I	SOURCE	STATEMENT
1	1					NAME	SAMPLE
2	2			M		MAC1	MACRO
3	3			M		LOCAL	LLAB ; (1)
4	4			M		LLAB:	
5	5			M		BR	\$LLAB ; (2)
6	6			M		ENDM	
7	7						
8	8	000000				REF1: MAC1	; (3)
	9			#1		;	
	10	000000		#1		??RA0000:	
	11	000000	14FE	#1		BR	\$\$\$RA0000 ; (2)
9	12						
10	13	000002	2C0000			BR	!LLAB ; (4)
***	ERROR	F407,	STNO	13	(0)	Undefined symbol reference 'LLAB'
***	ERROR	F303,	STNO	13	(13)	Illegal expression
11	14						
12	15	000005				REF2: MAC1	; (5)
	16			#1		;	
	17	000005		#1		??RA0001:	
	18	000005	14FE	#1		BR	\$\$\$RA0001 ; (2)
13	19						
14	20					END	

REPT

repeat

REPT

(3) REPT (repeat)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	REPT	absolute-expression	[:comment]
	:		
	ENDM		[:comment]

[Function]

- The REPT directive tells the assembler to repeatedly expand a series of statements described between this directive and the ENDM directive (called the REPT-ENDM block) the number of times equivalent to the value of the expression specified in the operand field.

[Use]

- Use the REPT and ENDM directives to describe a series of statements repeatedly in a source program.

[Explanation]

- An error occurs if the REPT directive is not paired with the ENDM directive.
- In the REPT-ENDM block, macro references, REPT directives, and IRP directives can be nested up to eight levels.
- If the EXITM directive appears in the REPT-ENDM block, subsequent expansion of the REPT-ENDM block by the assembler is terminated.
- Assembly control instructions may be described in the REPT-ENDM block.
- Macro definitions cannot be described in the REPT-ENDM block.
- The absolute expression described in the operand field is evaluated with unsigned 24 bits. If the value of the expression is 0, nothing is expanded.

REPT

repeat

REPT

[Application example]

<Source program>

NAME	SAMP1	
CSEG		
REPT	3	; (1)
INC	B	
DEC	C	
ENDM		; (2)
END		

← REPT-ENDM block

<Explanation>

- (1) This REPT directive tells the assembler to expand the REPT-ENDM block three consecutive times.
- (2) This directive indicates the end of the REPT-ENDM block.

When the above source program is assembled, the REPT-ENDM block is expanded as shown in the following assemble list.

<Assemble list>

NAME	SAMP1
CSEG	
REPT	3
INC	B
DEC	C
ENDM	
INC	B
DEC	C
INC	B
DEC	C
INC	B
DEC	C
END	

The REPT-ENDM block defined by statements (1) and (2) has been expanded three times. On the assemble list, the definition statements (1) and (2) by the REPT directive in the source module is not displayed.

IRP

indefinite repeat

IRP

(4) IRP (indefinite repeat)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	IRP	formal-parameter, <[actual- parameter [...]]>	[:comment]
	⋮		
	ENDM		[:comment]

[Function]

- The IRP directive tells the assembler to repeatedly expand a series of statements described between this directive and the ENDM directive (called the IRP-ENDM block) the number of times equivalent to the number of actual parameters while replacing the formal parameter with the actual parameters specified in the operand field.

[Use]

- Use the IRP and ENDM directives to describe a series of statements, only some of which become variables, repeatedly in a source program.

[Explanation]

- The IRP directive must be paired with the ENDM directive.
- Up to 16 actual parameters may be described in the operand field.
- In the IRP-ENDM block, macro references, REPT and IRP directives can be nested up to eight levels.
- If the EXITM directive appears in the IRP-ENDM block, subsequent expansion of the IRP-ENDM block by the assembler is terminated.
- Macro definitions cannot be described in the IRP-ENDM block.
- Assembly control instructions may be described in the IRP-ENDM block.

IRP

indefinite repeat

IRP

[Application example]

<Source program>

NAME	SAMP1	
CSEG		
IRP	PARA, <0AH, 0BH, 0CH>	; (1)
ADD	A, #PARA	
MOV	[DE+], A	
ENDM		; (2)
END		

← IRP-ENDM block

<Explanation>

- (1) The formal parameter is "PARA" and the actual parameters are the following three: "0AH", "0BH", and "0CH". This IRP directive tells the assembler to expand the IRP-ENDM block three times (i.e., the number of actual parameters) while replacing the formal parameter "PARA" with the actual parameters "0AH", "0BH", and "0CH".
- (2) This directive indicates the end of the IRP-ENDM block.

When the above source program is assembled, the IRP-ENDM block is expanded as shown in the following assemble list.

<Assemble list>

NAME	SAMP1	
CSEG		
ADD	A, #0AH	; (3)
MOV	[DE+], A	
ADD	A, #0BH	; (4)
MOV	[DE+], A	
ADD	A, #0CH	; (5)
MOV	[DE+], A	
END		

The IRP-ENDM block defined by statements (1) and (2) has been expanded three times (equivalent to the number of actual parameters).

- (3) In this ADD instruction, PARA is replaced with 0AH.
- (4) In this ADD instruction, PARA is replaced with 0BH.
- (5) In this ADD instruction, PARA is replaced with 0CH.

EXITM

exit from macro

EXITM

(5) EXITM (exit from macro)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	EXITM	None	[:comment]

[Function]

- The EXITM directive forcibly terminates the expansion of the macro body defined by the MACRO directive and the repetition by the REPT-ENDM or IRP-ENDM block.

[Use]

- This function is mainly used when a conditional assembly function (see **4.7 Conditional Assembly Control Instructions**) is used in the macro body defined with the MACRO directive.
- If conditional assembly functions are used in combination with other instructions in the macro body, part of the source program that must not be assembled is likely to be assembled unless control is returned from the macro by force using this EXITM directive. In such cases, be sure to use the EXITM directive.

[Explanation]

- If the EXITM directive is described in a macro body, instructions up to the ENDM directive will be stored as the macro body.
- The EXITM directive indicates the end of a macro only during the macro expansion.
- If something is described in the operand field of the EXITM directive, the assembler will output an error message but will execute the EXITM processing.
- If the EXITM directive appears in a macro body, the assembler will return by force the nesting level of IF/_IF/ELSE/ELSEIF/_ELSEIF/ENDIF blocks to the level when the assembler entered the macro body.
- If the EXITM directive appears in an INCLUDE file resulting from expanding the INCLUDE control instruction described in a macro body, the assembler will accept the EXITM directive as valid and terminate the macro expansion at that level.

EXITM

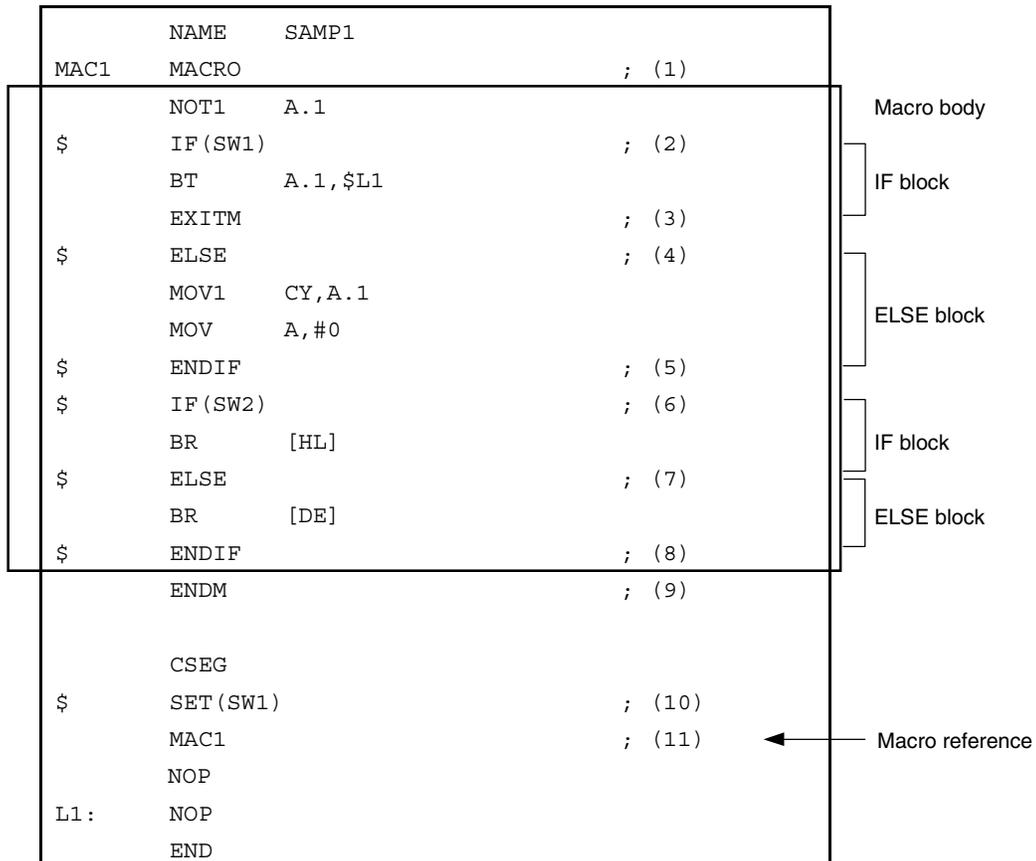
exit from macro

EXITM

[Application example]

- In the example here, conditional assembly control instructions are used. See **4.7 Conditional Assembly Control Instructions**.
- See **CHAPTER 5 MACROS** for the macro body and macro expansion.

<Source program>



EXITM

exit from macro

EXITM

<Explanation>

- (1) The macro "MAC1" uses conditional assembly functions (2) and (4) through (8) within the macro body.
- (2) An IF block for conditional assembly is defined here. If the switch name "SW1" is true (not "0"), the ELSE block is assembled.
- (3) This directive terminates by force the expansion of the macro body in (4) and thereafter. If this EXITM directive is omitted, the assembler proceeds to the assembly process in (6) and thereafter when the macro is expanded.
- (4) An ELSE block for conditional assembly is defined here. If the switch name "SW1" is false ("0"), the ELSE block is assembled.
- (5) This ENDIF control instruction indicates the end of the conditional assembly.
- (6) Another IF block for conditional assembly is defined here. If the switch name "SW2" is true (not "0"), the following IF block is assembled.
- (7) Another ELSE block for conditional assembly is defined. If the switch name "SW2" is false ("0"), the ELSE block is assembled.
- (8) This ENDIF instruction indicates the end of the conditional assembly processes in (6) and (7).
- (9) This directive indicates the end of the macro body.
- (10) This SET control instruction gives true value (not "0") to the switch name "SW1" and sets the condition of the conditional assembly.
- (11) This macro reference calls the macro "MAC1".

When the source program in the above example is assembled, macro expansion occurs as shown below.

NAME	SAMP1		
MAC1	MACRO		; (1)
	:		
	ENDM		; (9)
	CSEG		
\$	SET (SW1)		; (10)
	MAC1		; (11)
	NOT1	CY	
\$	IF (SW1)		
	BT	A.1, \$L1	
L1:	NOP		
	NOP		
	END		

Macro-expanded part

The macro body of the macro "MAC1" is expanded by referring to the macro in (11). Because true value is set in the switch name "SW1" in (10), the first IF block in the macro body is assembled. Because the EXITM directive is described at the end of the IF block, the subsequent macro expansion is not executed.

ENDM

end macro

ENDM

(6) ENDM (end macro)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
None	ENDM	None	[:comment]

[Function]

- The ENDM directive instructs the assembler to terminate the execution of a series of statements defined as the functions of the macro.

[Use]

- The ENDM directive must always be described at the end of a series of statements following the MACRO, REPT, and/or the IRP directives.

[Explanation]

- A series of statements described between the MACRO directive and ENDM directive becomes a macro body.
- A series of statements described between the REPT directive and ENDM directive becomes a REPT-ENDM block.
- A series of statements described between the IRP directive and ENDM directive becomes an IRP-ENDM block.

[Application examples]**Example 1 <MACRO-ENDM>**

```

NAME      SAMP1
ADMAC MACRO  PARA1, PARA2
           MOV   A, #PARA1
           ADD   A, #PARA2
           ENDM
           :
           END

```

ENDM

end macro

ENDM

Example 2 <REPT-ENDM>

```
NAME    SAMP2
CSEG
:
REPT    3
INC     B
DEC     C
ENDM
:
END
```

Example 3 <IRP-ENDM>

```
NAME    SAMP3
CSEG
:
IRP     PARA, <1, 2, 3>
ADD     A, #PARA
MOV     [DE+], A
ENDM
:
END
```

3.10 Assembly Termination Directive

The assembly termination directive informs the assembler of the end of a source module. This assembly termination directive must always be described at the end of each source module.

The assembler processes a series of statements up to the assembly termination directive as a source module. Therefore, if the assembly termination directive exists before the ENDM in a REPT block or an IRP block, the REPT block or IRP block becomes invalid.

END

end

END

(1) END (end)**[Description format]**

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
None	END	None	[:comment]

[Function]

- The END directive indicates to the assembler the end of a source module.

[Use]

- The END directive must always be described at the end of each source module.

[Explanation]

- The assembler continues to assemble a source module until the END directive appears in the source module. Therefore, the END directive is required at the end of each source module.
- Always input a line-feed (LF) code after the END directive.
- If any statement other than blank, tab, LF, or comments appears after the END directive, the assembler outputs a warning message.

[Application Example]

```

NAME      SAMPLE
DSEG
:
CSEG
:
END                ; (1)

```

<Explanation>

- (1) Always describe the END directive at the end of each source module.

CHAPTER 4 CONTROL INSTRUCTIONS

This chapter explains the control instructions. Control instructions provide detailed directions on the operation of the assembler.

4.1 Overview of Control Instructions

Control instructions are described in a source program to provide detailed directions on the operation of the assembler.

These instructions are not subject to object code generation.

Control instructions are available in the following types.

Table 4-1. List of Control Instructions

No.	Type of Control Instruction	Control Instruction
1	Processor type specification control instruction	PROCESSOR
2	Debug information output control instructions	DEBUG/NODEBUG, DEBUGA/NODEBUGA
3	Cross-reference list output specification control instructions	XREF/NOXREF, SYMLIST/NOSYMLIST
4	Inclusion control instruction	INCLUDE
5	Assembly list control instructions	EJECT, TITLE, SUBTITLE, LIST/NOLIST, GEN/NOGEN, COND/NOCOND, FORMFEED/NOFORMFEED, WIDTH, LENGTH, TAB
6	Conditional assembly control instructions	SET/RESET, IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF
7	SFR area change control instructions	CHGSFR/CHGSFRA
8	Other control instructions	DGL, DGS, TOL_INF

Control instructions are described in a source program in the same way as the assembler directives.

Of the control instructions listed in **Table 4-1 List of Control Instructions**, the following instructions have the same functions as assembler options that can be specified in the start-up command line of the assembler.

The correspondence between the control instructions and the command line assembler options is given in **Table 4-2 Control Instructions and Assembler Options**.

Table 4-2. Control Instructions and Assembler Options

Control Instruction	Assembler Option
PROCESSOR	-C
DEBUG/NODEBUG	-G/-NG
DEBUGA/NODEBUGA	-GA/-NGA
XREF/NOXREF	-KX/-NKX
SYMLIST/NOSYMLIST	-KS/-NKS
FORMFEED/NOFORMFEED	-LF/-NLF
TITLE	-LH
WIDTH	-LW
LENGTH	-LL
TAB	-LT
CHGSFR/CHGSFRA	-CS/-CSA

For the method of specifying the control instructions and assembler options by command line, see the **RA78K0 Assembler Package Operation**.

4.2 Processor Type Specification Control Instruction

The processor type specification control instruction specifies in a source module file the type of device subject to assembly.

PROCESSOR

processor

PROCESSOR

(1) PROCESSOR (processor)**[Description format]**

```
[Δ] $ [Δ] PROCESSOR [Δ] ( [Δ] processor-type [Δ] )
```

```
[Δ] $ [Δ] PC [Δ] ( [Δ] processor-type [Δ] )
```

; Abbreviated format

[Function]

- The PROCESSOR control instruction specifies in a source module file the processor type of the device subject to assembly.

[Use]

- The processor type of the device subject to assembly must always be specified in the source module file or in the start-up command line of the assembler.
- If the processor type specification for the device subject to assembly is omitted in each source module file, the processor type must be specified at each assembly operation. Therefore, by specifying the target device subject to assembly in each source module file, you can save time and trouble when starting up the assembler.

[Explanation]

- The PROCESSOR control instruction can be described only in the header section of a source module file. If the control instruction is described elsewhere, the assembler will be aborted.
- If the specified processor type differs from the actual target device subject to assembly, the assembler will be aborted.
- Only one PROCESSOR control instruction can be specified in the module header.
- The processor type of the target device subject to assembly may also be specified with the assembler option (-C) in the start-up command line of the assembler. If the specified processor type differs between the source module file and the start-up command line, the assembler will output a warning message and give precedence to the processor type specification in the start-up command line.
- Even when the assembler option (-C) has been specified in the start-up command line, the assembler performs a syntax check on the PROCESSOR control instruction.
- If the processor type is not specified in either the source module file or the start-up command line, the assembler will be aborted.

[Application example]

```
$      PROCESSOR (4038)
$      DEBUG
$      XREF

      NAME          TEST

      CSEG
      :
```

4.3 Debug Information Output Control Instructions

The debug information output control instructions are used to specify in a source module file the output or non-output of debugging information to an object module file created from the source module file.

DEBUG/NODEBUG

debug/nodebug

DEBUG/NODEBUG

(1) DEBUG/NODEBUG (debug/nodebug)**[Description format]**

[Δ] \$ [Δ] DEBUG

; Default assumption

[Δ] \$ [Δ] NODEBUG

[Function]

- The DEBUG control instruction tells the assembler to add local symbol information to an object module file.
- The NODEBUG control instruction tells the assembler not to add local symbol information to an object module file. However, in this case as well, the segment name is output to an object module file.
- “Local symbol information” refers to symbols other than module names and PUBLIC, EXTRN, and EXTBIT symbols.

[Use]

- Use the DEBUG control instruction when symbolic debugging including local symbols is to be performed.
- Use the NODEBUG control instruction when:
 1. Symbolic debugging is to be performed for global symbols only
 2. Debugging is to be performed without symbols
 3. Only objects are required (as for evaluation with PROM)

[Explanation]

- The DEBUG or NODEBUG control instruction can be described only in the header section of a source module file.
- If the DEBUG or NODEBUG control instruction is omitted, the assembler will assume that the DEBUG control instruction has been specified.
- The addition of local symbol information can be specified using the assembler option (-G/-NG) in the start-up command line.
- If the control instruction specification in the source module file differs from the specification in the start-up command line, the specification in the command line takes precedence.
- Even when the assembler option (-NG) has been specified, the assembler performs a syntax check on the DEBUG or NODEBUG control instruction.

DEBUGA/NODEBUGA

debuga/nodebuga

DEBUGA/NODEBUGA

(2) DEBUGA/NODEBUGA (debuga/nodebuga)**[Description format]**

[Δ] \$ [Δ] DEBUGA	; Default assumption
[Δ] \$ [Δ] NODEBUGA	

[Function]

- The DEBUGA control instruction tells the assembler to add assembler source debugging information to an object module file.
- The NODEBUGA control instruction tells the assembler not to add assembler source debugging information to an object module file.

[Use]

- Use the DEBUGA control instruction when debugging is to be performed at the assembler or structured assembler source level. An integrated debugger will be necessary for debugging at the source level.
- Use the NODEBUGA control instruction when:
 1. Debugging is to be performed without the assembler source
 2. Only objects are required (as for evaluation with PROM)

[Explanation]

- The DEBUGA or NODEBUGA control instruction can be described only in the header section of a source module file.
- If the DEBUGA or NODEBUGA control instruction is omitted, the assembler will assume that the DEBUGA control instruction has been specified.
- If two or more of these control instructions are specified, the last specified control instruction takes precedence over the others.
- The addition of assembler source debugging information can be specified using the assembler option (-GA/-NGA) in the start-up command line.
- If the control instruction specification in the source module file differs from the specification in the start-up command line, the specification in the command line takes precedence.
- Even when the assembler option (-NGA) has been specified, the assembler performs a syntax check on the DEBUGA or NODEBUGA control instruction.
- If compiling or structure-assembling the debug information output by the C compiler or structured assembler preprocessor, do not describe the debug information output control instructions when assembling the output assemble source. The control instructions necessary at assembly are output to assembler source as control statements by the C compiler or structured assembler preprocessor.

4.4 Cross-Reference List Output Specification Control Instructions

The cross-reference list output specification control instructions are used in a source module file to specify the output or non-output of a cross-reference list.

XREF/NOXREF

xref/noxref

XREF/NOXREF

(1) XREF/NOXREF (xref/noxref)**[Description format]**

[Δ] \$ [Δ] XREF	
[Δ] \$ [Δ] XR	; Abbreviated format
[Δ] \$ [Δ] NOXREF	; Default assumption
[Δ] \$ [Δ] NOXR	; Abbreviated format

[Function]

- The XREF control instruction tells the assembler to output a cross-reference list to an assembly list file.
- The NOXREF control instruction tells the assembler not to output a cross-reference list to an assembly list file.

[Use]

- Use the XREF control instruction to output a cross-reference list when you want information on where each of the symbols defined in the source module file is referenced or how many such symbols are referenced in the source module file.
- If you must specify the output or non-output of a cross-reference list at each assembly operation, you may save time and labor by specifying the XREF and NOXREF control instruction in the source module file.

[Explanation]

- The XREF or NOXREF control instruction can be described only in the header section of a source module file.
- If two or more of these control instructions are specified, the last specified control instruction takes precedence over the others.
- Output or non-output of a cross-reference list can also be specified by the assembler option (-KX/-NKX) in the start-up command line.
- If the control instruction specification in the source module file differs from the assembler option specification in the start-up command line, the specification in the command line will take precedence over that in the source module.
- Even when the assembler option (-NP) has been specified in the start-up command line, the assembler performs a syntax check on the XREF/NOXREF control instruction.

SYMLIST/NOSYMLIST

symlist/nosymlist

SYMLIST/NOSYMLIST

(2) SYMLIST/NOSYMLIST (symlist/nosymlist)**[Description format]**

[Δ] \$ [Δ] SYMLIST

[Δ] \$ [Δ] NOSYMLIST

; Default assumption

[Function]

- The SYMLIST control instruction tells the assembler to output a symbol list to a list file.
- The NOSYMLIST control instruction tells the assembler not to output a symbol list to a list file.

[Use]

- Use the SYMLIST control instruction to output a symbol list.

[Explanation]

- The SYMLIST or NOSYMLIST control instruction can be described only in the header section of a source module file.
- If two or more of these control instructions are specified, the last specified control instruction takes precedence over the others.
- Output of a symbol list can also be specified by the assembler option (-KS/-NKS) in the start-up command line.
- If the control instruction specification in the source module file differs from the assembler option specification in the start-up command line, the specification in the command line will take precedence over that in the source module.
- Even when the assembler option (-NP) has been specified in the start-up command line, the assembler performs a syntax check on the SYMLIST/NOSYMLIST control instruction.

4.5 Inclusion Control Instruction

The inclusion control instruction is used in a source module file to specify the inclusion of another module file in the source module file.

By making effective use of this control instruction, you can save time and labor in describing a source program.

INCLUDE

include

INCLUDE

(1) INCLUDE (include)**[Description Format]**

[Δ] \$ [Δ] INCLUDE [Δ] ([Δ] filename [Δ])
[Δ] \$ [Δ] IC [Δ] ([Δ] filename [Δ]) ; Abbreviated format

[Function]

- The INCLUDE control instruction tells the assembler to insert and expand the contents of a specified file beginning on a specified line in the source program for assembly.

[Use]

- A relatively large group of statements that may be shared by two or more source modules should be combined into a single file as an INCLUDE file. If the group of statements must be used in each source module, specify the filename of the required INCLUDE file with the INCLUDE control instruction. With this control instruction, you can greatly reduce time and labor in describing source modules.

[Explanation]

- The INCLUDE control instruction can only be described in ordinary source programs.
- The pathname or drive name of an INCLUDE file can be specified with the assembler option (-I).
- The assembler searches INCLUDE file read paths in the following sequence.
 - (a) When an INCLUDE file is specified without pathname specification
 - <1> Path in which the source file exists
 - <2> Path specified by the assembler option (-I)
 - <3> Path specified by the environment variable INC78K4
 - (b) When an INCLUDE file is specified with a pathname

If the INCLUDE file is specified with a drive name or a pathname beginning with (\), the path specified with the INCLUDE file will be prefixed to the INCLUDE filename. If the INCLUDE file is specified with a relative path (which does not begin with (\)), a pathname will be prefixed to the INCLUDE filename in the order described in (a) above.
- Nesting of INCLUDE files is allowed up to seven levels. In other words, the nesting level display of INCLUDE files in the assembly list is up to 8 (the term “nesting” here refers to the specification of one or more other INCLUDE files in an INCLUDE file).
- The END directive need not be described in an INCLUDE file.
- If the specified INCLUDE file cannot be opened, the assembler will abort operation.

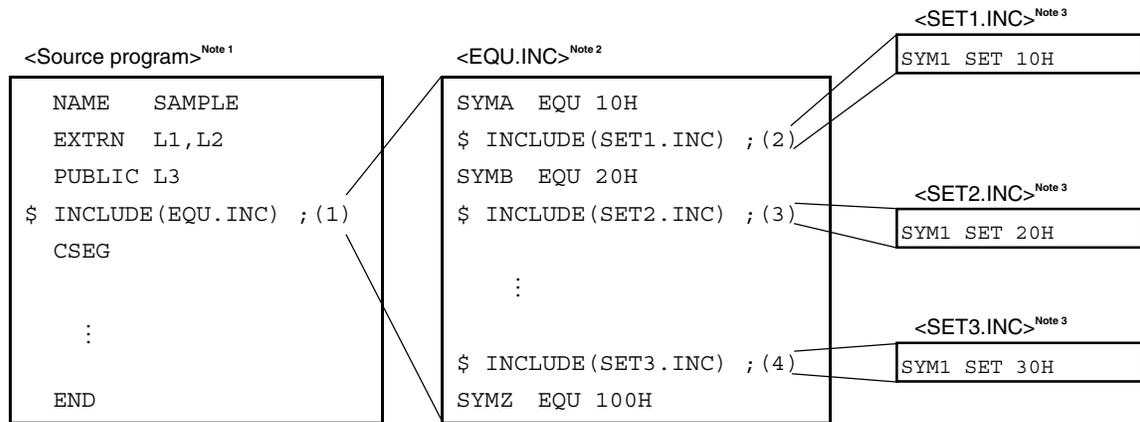
INCLUDE

include

INCLUDE

- An INCLUDE file must be closed with an IF or _IF control instruction that is properly paired with an ENDIF control instruction within the INCLUDE file. If the IF level at the entry of the INCLUDE file expansion does not correspond with the IF level immediately after the INCLUDE file expansion, the assembler will output an error message and force the IF level to return to that level at the entry of the INCLUDE file expansion.
- When defining a macro in an INCLUDE file, the macro definition must be closed in the INCLUDE file. If an ENDM directive appears unexpectedly (without the corresponding MACRO directive) in the INCLUDE file, an error message will be output and the ENDM directive will be ignored. If an ENDM directive is missing for the MACRO directive described in the INCLUDE file, the assembler will output an error message but will process the macro definition by assuming that the corresponding ENDM directive has been described.

[Application example]



- Notes**
1. Two or more \$IC control instructions can be specified in the source file. The same INCLUDE file may also be specified more than once.
 2. Two or more \$IC control instructions may be specified for INCLUDE file "EQU.INC".
 3. No \$IC control instruction can be specified in any of the INCLUDE files "SET1.INC", "SET2.INC", and "SET3.INC".

<Explanation>

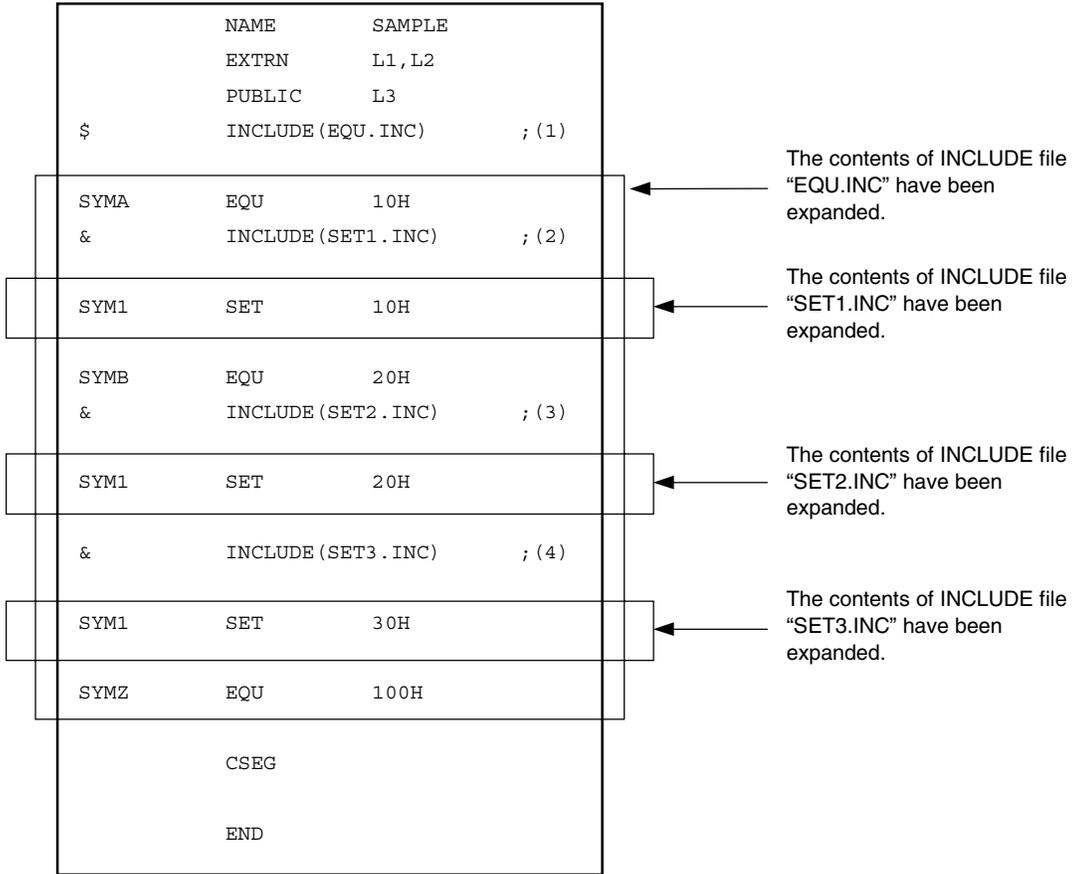
- (1) This control instruction specifies "EQU.INC" as the INCLUDE file.
- (2), (3), (4) These control instructions specify "SET1.INC", "SET2.INC", and "SET3.INC" as the INCLUDE file.

When this source program is assembled, the contents of the INCLUDE file will be expanded as follows.

INCLUDE

include

INCLUDE



4.6 Assembly List Control Instructions

The assembly list control instructions are used in a source module file to control the output format of an assembly list such as page ejection, suppression of list output, and subtitle output.

The assembly list control instructions include:

- EJECT
- LIST and NOLIST
- GEN and NOGEN
- COND and NOCOND
- TITLE
- SUBTITLE
- FORMFEED and NOFORMFEED
- WIDTH
- LENGTH
- TAB

EJECT

eject

EJECT

(1) EJECT (eject)**[Description format]**

[Δ] \$ [Δ] EJECT

[Δ] \$ [Δ] EJ

; Abbreviated format

[Default assumption]

- EJECT control instruction is not specified.

[Function]

- The EJECT control instruction causes the assembler to execute page ejection (formfeed) of an assembly list.

[Use]

- Describe the EJECT control instruction in a line of the source module at which page ejection of the assembly list is required.

[Explanation]

- The EJECT control instruction can only be described in ordinary source programs.
- Page ejection of the assembly list is executed after the image of the EJECT control instruction itself is output.
- If the assembler option (-NP) or (-LLO) is specified in the start-up command line or if the assembly list output is disabled by another control instruction, the EJECT control instruction becomes invalid. See the **RA78K4 Assembler Package Operation** for those assembler options.
- If an illegal description follows the EJECT control instruction, the assembler will output an error message.

EJECT

eject

EJECT

[Application example]

<Source module>

```

      :
      MOV      [DE+], A
      BR       $$
$     EJECT           ; (1)
      CSEG
      :
      END

```

<Explanation>

- (1) When page ejection is executed with the EJECT control instruction, the output assembly list will look like this.

```

      :
      MOV      [DE+], A
      BR       $$
$     EJECT
----- Page ejection
      CSEG
      :
      END

```

LIST/NOLIST

list/nolist

LIST/NOLIST

(2) LIST/NOLIST (list/nolist)**[Description format]**

[Δ] \$ [Δ] LIST	; Default assumption
[Δ] \$ [Δ] LI	; Abbreviated format
[Δ] \$ [Δ] NOLIST	
[Δ] \$ [Δ] NOLI	; Abbreviated format

[Function]

- The LIST control instruction indicates to the assembler the line at which assembly list output must start.
- The NOLIST control instruction indicates to the assembler the line at which assembly list output must be suppressed.

All source statements described after the NOLIST control instruction specification will be assembled, but will not be output on the assembly list until the LIST control instruction appears in the source program.

[Use]

- Use the NOLIST control instruction to limit the amount of assembly list output.
- Use the LIST control instruction to cancel the assembly list output suppression specified by the NOLIST control instruction.

By using a combination of NOLIST and LIST control instructions, you can control the amount of assembly list output as well as the contents of the list.

[Explanation]

- The LIST/NOLIST control instruction can only be described in ordinary source programs.
- The NOLIST control instruction functions to suppress assembly list output and is not intended to stop the assembly process.
- If the LIST control instruction is specified after the NOLIST control instruction, statements described after the LIST control instruction will be output again on the assembly list. The image of the LIST or NOLIST control instruction will also be output on the assembly list.
- If neither the LIST nor NOLIST control instruction is specified, all statements in the source module will be output to an assembly list.

LIST/NOLIST

list/nolist

LIST/NOLIST

[Application example]

	NAME	SAMP1	
\$	NOLIST		; (1)
DATA1	EQU	10H	
DATA2	EQU	11H	
	:		
DATA3	EQU	20H	
DATA4	EQU	20H	
\$	LIST		; (2)
	CSEG		
	:		
	END		

↑
 Statements in this part will
 not be output to the
 assembly list.
 ↓

<Explanation>

- (1) Because the NOLIST control instruction is specified here, statements after "\$ NOLIST" and up to the LIST control instruction in (2) will not be output on the assembly list. The image of the NOLIST control instruction itself will be output on the assembly list.
- (2) Because the LIST control instruction is specified here, statements after this control instruction will be output again on the assembly list. The image of the LIST control instruction itself will also be output on the assembly list.

GEN/NOGEN

generate/no generate

GEN/NOGEN

(3) GEN/NOGEN (generate/no generate)**[Description format]**

[Δ] \$ [Δ] GEN

; Default assumption

[Δ] \$ [Δ] NOGEN

[Function]

- The GEN control instruction tells the assembler to output macro definition lines, macro reference lines, and macro-expanded lines to an assembly list.
- The NOGEN control instruction tells the assembler to output macro definition lines and macro reference lines but to suppress macro-expanded lines.

[Use]

- Use the GEN/NOGEN control instruction to limit the amount of assembly list output.

[Explanation]

- The GEN/NOGEN control instruction can only be described in ordinary source programs.
- If neither the GEN nor NOGEN control instruction is specified, macro definition lines, macro reference lines, and macro-expanded lines will be output to an assembly list.
- The specified list control takes place after the image of the GEN or NOGEN control instruction itself has been printed on the assembly list.
- The assembler continues its processing and increments the statement number (STNO) count even after the list output control by the NOGEN control instruction.
- If the GEN control instruction is specified after the NOGEN control instruction, the assembler will resume the output of macro-expanded lines.

GEN/NOGEN

generate/no generate

GEN/NOGEN

[Application example]

<Source program>

	NAME	SAMP
\$	NOGEN	
ADMAC	MACRO	PARA1, PARA2
	MOV	A, #PARA1
	ADD	A, #PARA2
	ENDM	
	CSEG	
ADMAC		10H, 20H
	END	

When the above source program is assembled, the output assembly list will look like this.

	NAME	SAMP
\$	NOGEN	
ADMAC	MACRO	PARA1, PARA2
	MOV	A, #PARA1
	ADD	A, #PARA2
	ENDM	
	CSEG	
ADMAC		10H, 20H
	MOV	A, #10H
	AUD	A, #20H
	END	

Macro-expanded part will not be output.

<Explanation>

- (1) Because the NOGEN control instruction is specified, the macro-expanded lines will not be output to the assembly list.

COND/NOCOND

condition/no condition

COND/NOCOND

(4) COND/NOCOND (condition/no condition)**[Description format]**

[Δ] \$ [Δ] COND

; Default assumption

[Δ] \$ [Δ] NOCOND

[Function]

- The COND control instruction tells the assembler to output lines that have satisfied the conditional assembly condition as well as those which have not satisfied the conditional assembly condition to an assembly list.
- The NOCOND control instruction tells the assembler to output only lines that have satisfied the conditional assembly condition to an assembly list. The output of lines that have not satisfied the conditional assembly condition and lines in which IF/_IF, ELSEIF/_ELSEIF, ELSE, and ENDIF have been described will be suppressed.

[Use]

- Use the COND/NOCOND control instruction to limit the amount of assembly list output.

[Explanation]

- The COND/NOCOND control instruction can only be described in ordinary source programs.
- If neither the COND nor NOCOND control instruction is specified, the assembler will output lines that have satisfied the conditional assembly condition as well as those which have not satisfied the conditional assembly condition to an assembly list.
- The specified list control takes place after the image of the COND or NOCOND control instruction itself has been printed on the assembly list.
- The assembler increments the ALNO and STNO counts even after the list output control by the NOCOND control instruction.
- If the COND control instruction is specified after the NOCOND control instruction, the assembler will resume the output of lines that have not satisfied the conditional assembly condition and lines in which IF/_IF, ELSEIF/_ELSEIF, ELSE, and ENDIF have been described.

COND/NOCOND

condition/no condition

COND/NOCOND

[Application example]

<Source program>

```
NAME      SAMP
$         NOCOND
$         SET (SW1)
$         IF (SW1)
MOV       A, #1H
$         ELSE
MOV       A, #0H
ENDIF
END
```

This part, though assembled, will not be output to the assembly list.

TITLE

title

TITLE

(5) TITLE (title)**[Description format]**

[Δ] \$ [Δ] TITLE [Δ] ([Δ] 'title-string' [Δ])

[Δ] \$ [Δ] TT [Δ] ([Δ] 'title-string' [Δ])
--

; Abbreviated format

[Default assumption]

- When the TITLE control instruction is not specified, the TITLE column of the assembly list header is left blank.

[Function]

- The TITLE control instruction specifies the character string to be printed in the TITLE column at each page header of an assembly list, symbol table list, or cross-reference list.

[Use]

- Use the TITLE control instruction to print a title on each page of a list so that the contents of the list can be easily identified.
- If you need to specify a title with the assembler option at each assembly time, you can save time and labor in starting the assembler by describing this control instruction in the source module file.

[Explanation]

- The TITLE control instruction can be described only in the header section of a source module file.
- If two or more TITLE control instructions are specified at the same time, the assembler will accept only the last specified TITLE control instruction as valid.
- Up to 60 characters can be specified as the title string. If the specified title string consists of 61 or more characters, the assembler will accept only the first 60 characters of the string as valid. However, if the character length specification per line of an assembly list file (a quantity “X”) is 119 characters or less, “X – 60 characters” will be acceptable.
- If a single quotation mark (') is to be used as part of the title string, describe the single quotation mark twice in succession.
- If no title string is specified (the number of characters in the title string = 0), the assembler will leave the TITLE column blank.
- If any character not included in **2.2.2 Character set** is found in the specified title string, the assembler will output “!” in place of the illegal character in the TITLE column.
- A title for an assembly list can also be specified with the assembler option (-LH) in the start-up command line of the assembler.

TITLE

title

TITLE

[Application example]

<Source module>

```
$    PROCESSOR(4038)
$    TITLE('THIS IS TITLE')
      NAME      SAMPLE
      $         EJECT
      CSEG
      END
```

When the above source program is assembled, the output assembly list will appear as shown on the next page (with the number of lines per page specified as 72).

TITLE

title

TITLE

```
78K/IV Series Assembler Vx.xx THIS IS TITLE Date:xx xxx xxxx Page: 1
```

```
Command: sample.asm
```

```
Para-file:
```

```
In-file: SAMPLE.ASM
```

```
Obj-file: SAMPLE.REL
```

```
Prn-file: SAMPLE.PRN
```

```
Assemble list
```

ALNO	STNO	ADRS	OBJECT	M I	SOURCE STATEMENT
1	1			\$	PROCESSOR(4038)
2	2			\$	TITLE('THIS IS TITLE')
3	3				
4	4				NAME SAMP
5	5				
6	6			\$	EJECT

```
78K/IV Series Assembler Vx.xx THIS IS TITLE Date:xx xxx xxxx Page: 2
```

ALNO	STNO	ADRS	OBJECT	M I	SOURCE STATEMENT
7	7	-----			CSEG
8	8				
9	9				END
10	10				

```
Target chip : uPD784038
```

```
Device file : Vx.xx
```

```
Assembly complete, 0 error(s) and 0 warning(s) found. ( 0)
```

SUBTITLE

subtitle

SUBTITLE

(6) SUBTITLE (subtitle)**[Description format]**

```
[Δ] $ [Δ] SUBTITLE [Δ] ( [Δ] 'character-string' [Δ] )
```

```
[Δ] $ [Δ] ST [Δ] ( [Δ] 'character-string' [Δ] )
```

; Abbreviated format

[Default assumption]

- When the SUBTITLE control instruction is not specified, the SUBTITLE section of the assembly list header is left blank.

[Function]

- The SUBTITLE control instruction specifies the character string to be printed in the SUBTITLE section at each page header of an assembly list.

[Use]

- Use the SUBTITLE control instruction to print a subtitle on each page of an assembly list so that the contents of the assembly list can be easily identified. The character string of a subtitle may be changed for each page.

[Explanation]

- The SUBTITLE control instruction can only be described in ordinary source programs.
- Up to 72 characters can be specified as the subtitle string.
If the specified title string consists of 73 or more characters, the assembler will accept only the first 72 characters of the string as valid. A 2-byte character is counted as two characters, and tab is counted as one character.
- The character string specified with the SUBTITLE control instruction will be printed in the SUBTITLE section on the page after the page on which the SUBTITLE control instruction has been specified. However, if the control instruction is specified at the top (first line) of a page, the subtitle will be printed on that page.
- If the SUBTITLE control instruction has not been specified, the assembler will leave the SUBTITLE section blank.
- If a single quotation mark (') is to be used as part of the character string, describe the single quotation mark twice in succession.
- If the character string in the SUBTITLE section is 0, the SUBTITLE column will be left blank.
- If any character not included in **2.2.2 Character set** is found in the specified subtitle string, the assembler will output "!" in place of the illegal character in the SUBTITLE column. If CR (0DH) is described, an error will result and nothing will be output in the assembly list. If 00H is described, nothing from that point to the closing single quotation mark (') will be output.

SUBTITLE

subtitle

SUBTITLE

[Application example]

<Source module>

```
NAME      SAMP
CSEG
$  SUBTITLE('THIS IS SUBTITLE 1')      ; (1)
$  EJECT                               ; (2)
CSEG
$  SUBTITLE('THIS IS SUBTITLE 2')      ; (3)
$  EJECT                               ; (4)
$  SUBTITLE('THIS IS SUBTITLE 3')      ; (5)
END
```

<Explanation>

- (1) This control instruction specifies the character string 'THIS IS SUBTITLE 1'.
- (2) This control instruction specifies a page ejection.
- (3) This control instruction specifies the character string 'THIS IS SUBTITLE 2'.
- (4) This control instruction specifies a page ejection.
- (5) This control instruction specifies the character string 'THIS IS SUBTITLE 3'.

SUBTITLE

subtitle

SUBTITLE

The assembly list for this example appears as follows (with the number of lines per page specified as 80).

```

78K/IV Series Assembler Vx.xx                               Date:xx xxx xxxx Page:  1

    Assemble list

ALNO  STNO  ADRS   OBJECT      M I  SOURCE STATEMENT

    1     1                                     NAME   SAMP
    2     2
    3     3  -----                                CSEG
    4     4
    5     5                                     $      SUBTITLE('THIS IS SUBTITLE 1') ;(1)
    6     6                                     $      EJECT                               ;(2)
-----
78K/IV Series Assembler Vx.xx                               Date:xx xxx xxxx Page:  2

THIS IS SUBTITLE 1

ALNO  STNO  ADRS   OBJECT      M I  SOURCE STATEMENT

    7     7
    8     8  -----                                CSEG
    9     9
   10    10                                     $      SUBTITLE('THIS IS SUBTITLE 2') ;(3)
   11    11                                     $      EJECT                               ;(4)
-----
78K/IV Series Assembler Vx.xx                               Date:xx xxx xxxx Page:  3

THIS IS SUBTITLE 2

ALNO  STNO  ADRS   OBJECT      M I  SOURCE STATEMENT

   12    12
   13    13                                     $      SUBTITLE('THIS IS SUBTITLE 3') ;(5)
   14    14                                     END

Target chip : uPD784038
Device file  : Vx.xx
Assembly complete,      0 error(s) and  0 warning(s) found. (  0)

```

FORMFEED/NOFORMFEED

formfeed/noformfeed

FORMFEED/NOFORMFEED

(7) FORMFEED/NOFORMFEED (formfeed/noformfeed)**[Description format]**

[Δ] \$ [Δ] FORMFEED

[Δ] \$ [Δ] NOFORMFEED

; Default assumption

[Function]

- The FORMFEED control instruction tells the assembler to output a FORMFEED code at the end of an assembly list file.
- The NOFORMFEED control instruction tells the assembler not to output a FORMFEED code at the end of an assembly list file.

[Use]

- Use the FORMFEED control instruction when you want to start a new page after printing the contents of an assembly list file.

[Explanation]

- The FORMFEED or NOFORMFEED control instruction can be described only in the header section of a source module file.
- At the time of printing an assembly list, the last page of the list may not come out if printing ends in the middle of a page. In such a case, add a FORMFEED code to the end of the assembly list using the FORMFEED control instruction or assembler option (-LF).
In many cases, a FORMFEED code will be output at the end of a file. For this reason, if a FORMFEED code exists at the end of a list file, an unwanted white page may be ejected. To prevent this, the NOFORMFEED control instruction or assembler option (-NLF) has been set as a default value.
- If two or more FORMFEED/NOFORMFEED control instructions are specified at the same time, only the last specified control instruction will become valid.
- The output or non-output of a formfeed code may also be specified with the assembler option (-LF) or (-NLF) in the start-up command line of the assembler.
- If the control instruction specification (FORMFEED/NOFORMFEED) in the source module differs from the specification (-LF/-NLF) in the start-up command line, the specification in the start-up command line will take precedence over that in the source module.
- Even when the assembler option (-NP) has been specified in the start-up command line, the assembler performs a syntax check on the FORMFEED or NOFORMFEED control instruction.

WIDTH

width

WIDTH

(8) WIDTH (width)**[Description format]**

```
[Δ] $ [Δ] WIDTH [Δ] ( [Δ] columns-per-line [Δ] )
```

[Default assumption]

\$WIDTH (132)

[Function]

- The WIDTH control instruction specifies the number of columns (characters) per line of a list file. “columns-per-line” must be a value in the range of 72 to 250.

[Use]

- Use the WIDTH control instruction when you want to change the number of columns per line of a list file.

[Explanation]

- The WIDTH control instruction can be described only in the header section of a source module file.
- If two or more WIDTH control instructions are specified at the same time, only the last specified control instruction will become valid.
- The number of columns per line of a list file may also be specified with the assembler option (-LW) in the start-up command line of the assembler.
- If the control instruction specification (WIDTH) in the source module differs from the specification (-LW) in the start-up command line, the specification in the command line will take precedence over that in the source module.
- Even when the assembler option (-NP) has been specified in the start-up command line, the assembler performs a syntax check on the WIDTH control instruction.

LENGTH

length

LENGTH

(9) LENGTH (length)**[Description format]**

```
[Δ] $ [Δ] LENGTH [Δ] ( [Δ] lines-per-page [Δ] )
```

[Default assumption]

\$LENGTH (66)

[Function]

- The LENGTH control instruction specifies the number of lines per page of a list file. “lines-per-page” may be “0” or a value in the range of 20 to 32767.

[Use]

- Use the LENGTH control instruction when you want to change the number of lines per page of a list file.

[Explanation]

- The LENGTH control instruction can be described only in the header section of a source module file.
- If two or more LENGTH control instructions are specified at the same time, only the last specified control instruction will become valid.
- The number of columns per line of a list file may also be specified with the assembler option (-LL) in the start-up command line of the assembler.
- If the control instruction specification (LENGTH) in the source module differs from the specification (-LL) in the start-up command line, the specification in the command line will take precedence over that in the source module.
- Even when the assembler option (-NP) has been specified in the start-up command line, the assembler performs a syntax check on the LENGTH control instruction.

TAB

tab

TAB

(10) TAB (tab)**[Description format]**

[Δ] \$ [Δ] TAB [Δ] ([Δ] number-of-columns [Δ])
--

[Default assumption]

\$TAB (8)

[Function]

- The TAB control instruction specifies the number of columns as tab stops on a list file. “number-of-columns” may be a value in the range of 0 to 8.
- The TAB control instruction specifies the number of columns that becomes the basis of tabulation processing to output any list by replacing a HT (Horizontal Tabulation) code in a source module with several blank characters on the list.

[Use]

- Use HT code to reduce the number of blanks when the number of characters per line of any list is reduced using the TAB control instruction.

[Explanation]

- The TAB control instruction can be described only in the header section of a source module file.
- If two or more TAB control instructions are specified at the same time, only the last specified control instruction will become valid.
- The number of tab stops may also be specified with the assembler option (-LT) in the start-up command line of the assembler.
- If the control instruction specification (TAB) in the source module differs from the specification (-LT) in the start-up command line, the specification in the command line will take precedence over that in the source module.
- Even when the assembler option (-NP) has been specified in the start-up command line, the assembler performs a syntax check on the TAB control instruction.

4.7 Conditional Assembly Control Instructions

The conditional assembly control instructions are used to select a series of statements in a source module as those subject to assembly or not subject to assembly, by setting switches for conditional assembly.

These control instructions consist of the IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF control instructions and the SET/RESET control instructions.

By making effective use of these control instructions, you can assemble a source module that excludes unwanted statements, with little or no change to the source module.

IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF

IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF

(1) IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF**[Description format]**

```

[Δ] $ [Δ] IF [Δ] ( [Δ] switch-name [ [Δ] : [Δ] switch-name ] ... [Δ] )
or [Δ] $ [Δ] _IFΔconditional-expression
    :
[Δ] $ [Δ] ELSEIF [Δ] ( [Δ] switch-name [ [Δ] : [Δ] switch-name ] ... [Δ] )
or [Δ] $ [Δ] _ELSEIFΔconditional-expression
    :
[Δ] $ [Δ] ELSE
    :
[Δ] $ [Δ] ENDIF

```

[Function]

- The control instructions set the conditions to limit source statements subject to assembly. Source statements described between the IF or _IF control instruction and the ENDIF control instruction are subject to conditional assembly.
- If the evaluated value of the conditional expression or the switch name specified by the IF or _IF control instruction (i.e., IF or _IF condition) is true (other than 0000H), source statements described after this IF or _IF control instruction until the appearance of the next conditional assembly control instruction (ELSEIF/_ELSEIF, ELSE, or ENDIF) in the source program will be assembled. For subsequent assembly processing, the assembler will proceed to the statement next to the ENDIF control instruction. If the IF or _IF condition is false (0000H), source statements described after this IF or _IF control instruction until the appearance of the next conditional assembly control instruction (ELSEIF/_ELSEIF, ELSE, or ENDIF) in the source program will not be assembled.
- The ELSEIF or _ELSEIF control instruction is checked for true/false status only when the conditions of all the conditional assembly control instructions described before this ELSEIF or _ELSEIF control instruction are not satisfied (i.e. the evaluated values are false). If the evaluated value of the conditional expression or the switch name specified by the ELSEIF or _ELSEIF control instruction (i.e. ELSEIF or _ELSEIF condition) is true, source statements described after this ELSEIF or _ELSEIF control instruction until the appearance of the next conditional assembly control instruction (ELSEIF/_ELSEIF, ELSE, or ENDIF) in the source program will be assembled. For subsequent assembly processing, the assembler will proceed to the statement next to the ENDIF control instruction. If the ELSEIF or _ELSEIF condition is false, source statements described after this ELSEIF or _ELSEIF control instruction until the appearance of the next conditional assembly control instruction (ELSEIF/_ELSEIF, ELSE, or ENDIF) in the source program will not be assembled.
- If the conditions of all the IF/_IF and ELSEIF/_ELSEIF control instructions described before the ELSE control instruction are not satisfied (i.e., all the switch names are false), source statements described after this ELSE control instruction until the appearance of the ENDIF control instruction in the source program will be assembled.
- The ENDIF control instruction indicates to the assembler the termination of source statements subject to conditional assembly.

IF/ _IF/ELSEIF/ _ELSEIF/ELSE/ENDIF

IF/ _IF/ELSEIF/ _ELSEIF/ELSE/ENDIF

[Use]

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.
- If a statement for debugging that becomes necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

[Explanation]

- The IF and ELSEIF control instructions are used for true/false condition judgment with switch name(s), whereas the _IF and _ELSEIF control instructions are used for true/false condition judgment with a conditional expression.
Both IF/ELSEIF and _IF/_ELSEIF may be used in combination. In other words, ELSEIF/_ELSEIF may be used in a pair with IF or _IF and ENDIF.
- Describe absolute expression for a conditional expression.
- The rules of describing switch names are the same as the conventions of symbol description (for details, see **2.2.3 Fields that make up a statement**). However, the maximum number of characters that can be recognized as a switch name is always 8.
- If the two or more switch names are to be specified with the IF or ELSEIF control instruction, delimit each switch name with a colon (:). Up to five switch names can be used per module.
- When two or more switch names have been specified with the IF or ELSEIF control instruction, the IF or ELSEIF condition is judged to be satisfied if one of the switch name values is true.
- The value of each switch name to be specified with the IF or ELSEIF control instruction must be defined with the SET or RESET control instruction (see **4.7 (2) SET/RESET**). Therefore, if the value of the switch name specified with the IF or ELSEIF control instruction is not set in the source module with the SET or RESET control instruction in advance, it is assumed to be reset.
- If the specified switch name or conditional expression contains an illegal description, the assembler will output an error message and determine that the evaluated value is false.
- When describing the IF or _IF control instruction, the IF or _IF control instruction must always be paired with the ENDIF control instruction.
- If an IF-ENDIF block is described in a macro body and control is transferred back from the macro at that level by EXITM processing, the assembler will force the IF level to return to that level at the entry of the macro body. In this case, no error will result.
- Description of an IF-ENDIF block in another IF-ENDIF block is referred to as nesting of IF control instructions. Nesting of IF control instructions is allowed up to 8 levels.
- In conditional assembly, object codes will not be generated for statements not assembled, but these statements will be output without change on the assembly list. If you do not wish to output these statements, use the \$NOCOND control instruction.

[Application Examples]**Example 1**

```

    text0
$    IF (SW1)                ; (1)
    text1
$    ENDIF                  ; (2)
    :
    END

```

<Explanation>

- (1) If the value of switch name "SW1" is true, statements in "text1" will be assembled.
If the value of switch name "SW1" is false, statements in "text1" will not be assembled.
The value of switch name "SW1" has been set to true or false with the SET or RESET control instruction described in "text0".
- (2) This instruction indicates the end of the source statement range for conditional assembly.

Example 2

```

    text0
$    IF (SW1)                ; (1)
    text1
$    ELSE                  ; (2)
    text2
$    ENDIF                  ; (3)
    :
    END

```

<Explanation>

- (1) The value of switch name "SW1" has been set to true or false with the SET or RESET control instruction described in "text0".
If the value of switch name "SW1" is true, statements in "text1" will be assembled and statements in "text2" will not be assembled.
- (2) If the value of switch name "SW1" in (1) is false, statements in "text1" will not be assembled and statements in "text2" will be assembled.
- (3) This instruction indicates the end of the source statement range for conditional assembly.

Example 3

```

      text0
$      IF (SW1)                ; (1)
      text1
$      ELSEIF (SW2)           ; (2)
      text2
$      ELSEIF (SW3)           ; (3)
      text3
$      ELSE                    ; (4)
      text4
$      ENDIF                  ; (5)
      :
      END

```

<Explanation>

- (1) The values of the switch names "SW1", "SW2", and "SW3" have been set to true or false with the SET or RESET control instruction described in "text0".
If the value of the switch name "SW1" is true, statements in "text1" will be assembled and statements in "text2", "text3", and "text4" will not be assembled.
If the value of the switch name "SW1" is false, statements in "text1" will not be assembled and statements after (2) will be conditionally assembled.
- (2) If the value of the switch name "SW1" in (1) is false and the value of the switch name "SW2" is true, statements in "text2" will be assembled and statements in "text1", "text3", and "text4" will not be assembled.
- (3) If the values of both switch names "SW1" in (1) and "SW2" in (2) are false and the value of the switch name "SW3" is true, statements in "text3" will be assembled and statements in "text1", "text2", and "text4" will not be assembled.
- (4) If the values of switch names "SW1" in (1), "SW2" in (2), and "SW3" in (3) are all false, statements in "text4" will be assembled and statements in "text1", "text2", and "text3" will not be assembled.
- (5) This instruction indicates the end of the source statement range for conditional assembly.

Example 4

```

      text0
$      IF (SWA=SWB)           ; (1)
      text1
$      ENDIF                  ; (2)
      :
      END

```

<Explanation>

- (1) The values of the switch names "SWA" and "SWB" has been set to true or false with the SET or RESET control instruction described in "text0".
If the value of the switch name "SWA" or "SWB" is true, statements in "text1" will be assembled.
If the values of both switch names "SWA" and "SWB" are false, statements in "text1" will not be assembled.
- (2) This instruction indicates the end of the source statement range for conditional assembly.

(2) SET/RESET (set/reset)**[Description format]**

```
[Δ] $ [Δ] SET [Δ] ( [Δ] switch-name [ [Δ] : [Δ] switch-name] ... [Δ] )
[Δ] $ [Δ] RESET [Δ] ( [Δ] switch-name [ [Δ] : [Δ] switch-name] ... [Δ] )
```

[Function]

- The SET and RESET control instructions give a value to each switch name to be specified with the IF or ELSEIF control instruction.
- The SET control instruction gives a true value (00FFH) to each switch name specified in its operand.
- The RESET control instruction gives a false value (0000H) to each switch name specified in its operand.

[Use]

- Describe the SET control instruction to give a true value (00FFH) to each switch name to be specified with the IF or ELSEIF control instruction.
- Describe the RESET control instruction to give a false value (0000H) to each switch name to be specified with the IF or ELSEIF control instruction.

[Explanation]

- With the SET and RESET control instructions, at least one switch name must be described. The conventions for describing switch names are the same as the conventions for describing symbols (see **2.2.3 Fields that make up a statement**). However, the maximum number of characters that can be recognized as a switch name is always 31.
- The specified switch name(s) may be the same as user-defined symbol(s) other than reserved words and other switch names.
- If two or more switch names are to be specified with the SET or RESET control instruction, delimit each switch name with a colon (:). Up to 1,000 switch names can be used per module.
- A switch name once set to “true” with the SET control instruction can be changed to “false” with the RESET control instruction, and vice versa.
- A switch name to be specified with the IF or ELSEIF control instruction must be defined at least once with the SET or RESET control instruction in the source module before describing the IF or ELSEIF control instruction.
- Switch names will not be output to a cross-reference list.

[Application example]

```

$      SET (SW1)                ; (1)
      :
$      IF (SW1)                 ; (2)
      text1
$      ENDIF                    ; (3)
      :
$      RESET (SW1 : SW2)       ; (4)
      :
$      IF (SW1)                 ; (5)
      text2
$      ELSEIF (SW2)            ; (6)
      text3
$      ELSE                     ; (7)
      text4
$      ENDIF                    ; (8)
      :
      END

```

<Explanation>

- (1) This instruction gives a true value (00FFH) to the switch name "SW1".
- (2) Because the true value has been given to the switch name "SW1" in (1) above, statements in "text1" will be assembled.
- (3) This instruction indicates the end of the source statement range for conditional assembly that starts from (2).
- (4) This instruction gives a false value (0000H) to the switch names "SW1" and "SW2", respectively.
- (5) Because the false value has been given to the switch name "SW1" in (4) above, statements in "text2" will not be assembled.
- (6) Because the false value has also been given to the switch name "SW2" in (4) above, statements in "text3" will not be assembled.
- (7) Because both switch names "SW1" and "SW2" are false in (5) and (6) above, statements in "text4" will be assembled.
- (8) This instruction indicates the end of the source statement range for conditional assembly that starts from (5).

4.8 SFR Area Change Control Instructions

When a chip is ordered according to the end user's request, the SFR area and its vicinity (the relocatable space) can be changed. The SFR area change control instructions are control instructions provided to support the changes requested by the end user in the assembler and linker.

CHGSFR/CHGSFRA

change sfr area/change sfr area

CHGSFR/CHGSFRA

(1) CHGSFR/CHGSFRA (change sfr area/change sfr area)**[Description format]**

<p>[Δ] \$ [Δ] CHGSFR ([Δ] absolute-expression n [Δ]) [Δ] \$ [Δ] CHGSFRA</p>
--

[Default assumption]

- \$CHGSFR (0FH)

[Function]

- The CHGSFR control instruction specifies addresses in the SFR area.
When the operand has been specified as "0": 0FD00H to 0FFFFH
When the operand has been specified as "0FH": 0FFD00H to 0FFFFFFH
- The CHGSFRA control instruction instructs the assembler to check that no LOCATION instruction is described and that no location address of an absolute segment is found in the entire available SFR area.
Under normal circumstances, do not use this control instruction.

[Use]

- Describe this control instruction when you want to change the SFR area.

[Explanation]

- The CHGSFR or CHGSFRA control instruction can be described only in the header section of a source module file.
- If two or more of these control instructions are specified in the header section of a source module file, the last specified control instruction takes precedence over the others.
- Change of the SFR area can also be specified by the assembler option (-CS/-CSA) in the start-up command line.
- If the control instruction specification (CHGSFR/CHGSFRA) in the source module file differs from the assembler option specification (-CS/-CSA) in the start-up command line, the specification in the command line will take precedence over that in the source module.
- During linking, the specified values of all modules specified by the CHGSFR control instruction and the -CS assembler option must be the same.
That value must also be the same as the value specified by the LOCATION instruction.
- Even when the assembler option "-NO" has been specified in the start-up command line, the assembler performs a syntax check on the CHGSFR/CHGSFRA control instruction.

4.9 Other Control Instructions

The following control instructions are special control instructions output by high-level programs such as a C compiler and structured assembler preprocessor.

\$TOL_INF

\$DGS

\$DGL

CHAPTER 5 MACROS

This chapter explains how to use a macro function. A macro is a very useful function when you need to describe a series of statements repeatedly in a source program.

5.1 Overview of Macros

When you must describe a series or group of instructions repeatedly in a source program, a macro function is very useful for program description. The macro function refers to the expansion of a series of statements (an instruction group) defined as a macro body with the MACRO and ENDM directives at the location where the macro name is referenced.

A macro is used to increase the coding efficiency of a source program and is different from a subroutine.

Macros and subroutines have distinct features as explained below. For effective use, select either a macro or a subroutine according to the specific purpose.

(1) Subroutines

- Describe a process that must be repeated many times in a program as a single subroutine. The subroutine will be converted into machine language by the assembler only once.
- To call the subroutine, you only need to describe a subroutine call instruction (generally, instructions to set arguments are also described before and after the subroutine).
Effective use of subroutines enables program memory to be used with high efficiency.
- By coding a series of processes in a program as subroutines, the program can be structured (this structuring makes the overall structure of the program easy for the programmer to understand, making program design easy).

(2) Macros

- The basic function of a macro is the replacement of a group of instructions with a name.
A series (or group) of instructions defined as a macro body with the MACRO and ENDM directives will be expanded at the location where the macro name is referenced.
- When the assembler finds a macro reference, the assembler expands the macro body and converts the group of instructions into machine language while replacing the formal parameter(s) of the macro body with the actual parameters at the time of the macro reference.
- Parameters can be described for a macro.
For example, if there are instruction groups that are the same in processing procedure but are different in the data to be described in the operand, define a macro by assigning formal parameter(s) to the data. By describing the macro name and the actual parameter(s) when the macro is referenced, the assembler can cope with various instruction groups that differ only in part of the statement description.

Programming techniques using subroutines are mainly used to reduce memory size and structure programs, whereas macros are used to increase the coding efficiency of the program.

5.2 Utilization of Macros

5.2.1 Macro definition

A macro is defined with the MACRO and ENDM directives.

[Description format]

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
macro-name	MACRO	[formal-parameter [...]]	[:comment]
	⋮		
	ENDM		

[Function]

- The MACRO directive executes a macro definition by assigning the macro name specified in the symbol field to a series of statements (called a macro body) described between this directive and the ENDM directive.

[Application example]

ADMAC	MACRO	PARA1 , PARA2
	MOV	A, #PARA1
	ADD	A, #PARA2
	ENDM	

<Explanation>

The above example shows a simple macro definition that specifies the addition of two values “PARA1” and “PARA2” and the storage of the result in register A. The macro is given the name “ADMAC” and “PARA1” and “PARA2” are formal parameters.

For details, see **(1) MACRO (macro)** in **3.9 Macro Directives**.

5.2.2 Macro reference

To call a macro, the already defined macro name must be described in the mnemonic field of the source program.

[Description format]

<u>Symbol field</u>	<u>Mnemonic field</u>	<u>Operand field</u>	<u>Comment field</u>
[label:]	macro-name	[actual-parameter [...]]	[:comment]

[Function]

- This statement description calls the macro body assigned to the macro name specified in the mnemonic field.

[Use]

- Use this statement description to call a macro body.

[Explanation]

- The macro name to be specified in the mnemonic field must have been defined before the macro reference.
- Up to 16 actual parameters may be specified per line by delimiting each actual parameter with a comma (,).
- No blank can be described in the character string constituting an actual parameter.
- When describing a comma (,), semicolon (;), blank, or tab in an actual parameter, enclose the character string that includes any of these special characters with a pair of single quotation marks.
- Formal parameters are replaced with their corresponding actual parameters in sequence from left to right. A warning message will be output if the number of formal parameters is not equal to the number of actual parameters.

[Application example]

	NAME	SAMPLE
ADMAC	MACRO	PARA1, PARA2
	MOV	A, #PARA1
	ADD	A, #PARA2
	CSEG	
	:	
	ADMAC	10H, 20H
	:	
	END	

<Explanation>

This macro reference calls the already defined macro name "ADMAC". 10H and 20H are actual parameters.

5.2.3 Macro expansion

The assembler processes a macro as follows.

- The assembler expands the macro body corresponding to the referenced macro name at the location where the macro name is referenced.
- The assembler assembles statements in the expanded macro body in the same way as other statements.

[Application example]

When the macro referenced in **5.2.2 Macro reference** is assembled, the macro body will be expanded as shown below.

	NAME	SAMPLE	
	ADMAC	MACRO	PARA1 , PARA2
		MOV	A, #PARA1
		ADD	A, #PARA2
		ENDM	
		CSEG	
		:	
	ADMAC	10H, 20H	; (1)
		MOV	A, PARA1 10H
		ADD	A, PARA2 20H
		:	
		END	

Macro definition

Macro expansion

<Explanation>

By the macro reference in (1), the macro body will be expanded. The formal parameters within the macro body will be replaced with the actual parameters.

5.3 Symbols Within Macros

Symbols that can be defined in a macro are divided into two types: global symbols and local symbols.

(1) Global symbols

- A global symbol is a symbol that can be referenced from any statement within a source program. Therefore, if a macro in which the global symbol has been defined is referenced more than once to expand a series of statements, the symbol will cause a double definition error.
- Symbols not defined with the LOCAL directive are global symbols.

(2) Local symbols

- A local symbol is a symbol defined with the LOCAL directive (see **(2) LOCAL (local)** in **3.9 Macro Directives**).
- A local symbol can be referenced within the macro declared as LOCAL with the LOCAL directive.
- No local symbol can be referenced from outside the macro.

[Application example]

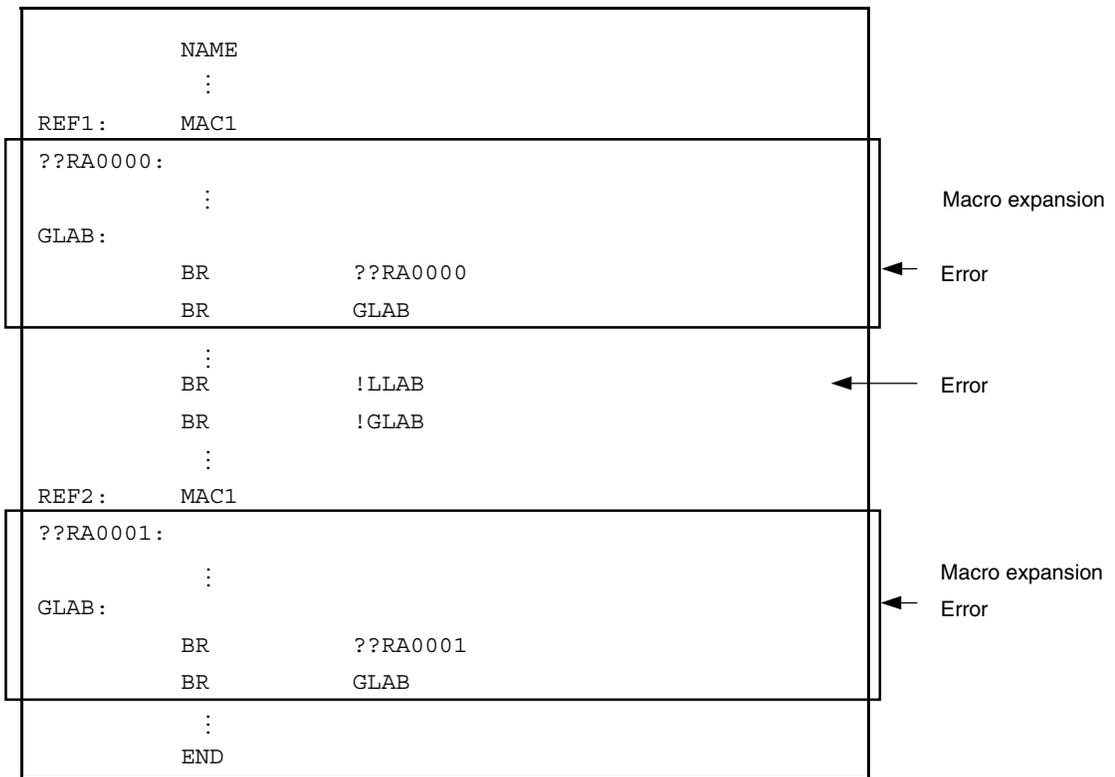
<Source program>

	NAME	SAMPLE	
MAC1	MACRO		
	LOCAL	LLAB	; (1)
LLAB:	:		
GLAB:			
	BR	LLAB	; (2)
	BR	GLAB	; (3)
	ENDM		
	:		
REF1:	MAC1		; (4) ← Macro reference
	:		
	BR	LLAB	; (5) ← This description is erroneous.
	BR	GLAB	; (6)
	:		
REF2:	MAC1		; (7) ← Macro reference
	:		
	END		

<Explanation>

- (1) This LOCAL directive defines the label “LLAB” as a local symbol.
- (2) This BR instruction references the local symbol “LLAB” in macro “MAC1”.
- (3) This BR instruction references the global symbol “GLAB” in macro “MAC1”.
- (4) This statement references the macro “MAC1”.
- (5) This BR instruction references the local symbol “LLAB” from outside the definition of the macro “MAC1”. This description causes an error when the source program is assembled.
- (6) This BR instruction references the global symbol “GLAB” from outside the definition of the macro “MAC1”.
- (7) This statement references the macro “MAC1”. The same macro is referenced twice.

When the source program in the above example is assembled, the macro body will be expanded as shown below.



<Explanation>

The global symbol "GLAB" has been defined in the macro "MAC1". Because the macro "MAC1" is referenced twice, the global symbol "GLAB" causes a double definition error as a result of expanding a series of statements in the macro body.

5.4 Macro Operators

Two types of macro operators are available: “& (ampersand)” and “” (single quotation mark)”.

(1) & (Concatenation)

- The ampersand “&” concatenates one character string to another within a macro body. When the macro is expanded, the character string on the left of the ampersand is concatenated to the character string on the right of the sign. The “&” itself disappears after concatenating the strings.
- When the macro is defined, a string before or after “&” in a symbol can be recognized as a formal parameter or LOCAL symbol. When the macro is expanded, the formal parameter or LOCAL symbol before or after “&” is evaluated as a symbol and can be concatenated in the symbol.
- The “&” sign enclosed in a pair of single quotation marks is simply handled as data.
- Two “&” signs described in succession are handled as a single “&” sign.

[Application example]

Macro definition

```

MAC      MACRO  P
LAB&P:
          D&B   10H
          DB    'P'
          DB    P
          DB    '&P'
          ENDM

```

← Formal parameter “P” is recognized.

Macro reference

```

          MAC   1H
LAB1H:
          DB   10H
          DB   'P'
          DB   1H
          DB   '&P'

```

← “D” and “B” are concatenated and become “DB”.

← & enclosed in a pair of single quotation marks is simply handled as data.

(2) ' (Single quotation mark)

- If a character string enclosed by a pair of single quotation marks is described at the beginning of an actual parameter in a macro reference line or an IRP directive or after a delimiting character, the character string will be interpreted as an actual parameter. The character string will be passed to the actual parameter without the enclosing single quotation marks.
- If a character string enclosed by a pair of single quotation marks exists in a macro body, the character string will simply be handled as data.
- To use a single quotation mark as a single quotation mark in text, describe the single quotation mark twice in succession.

[Application example]

```

NAME      SAMP
MAC1      MACRO  P
            IRP   Z, <P>
            MOV   A, #Z
            ENDM
            ENDM

            MAC1  '10,20,30'

```

When the source program in the above example is assembled, the macro "MAC1" will be expanded as shown below.

```

            IRP   Z, <10,20,30>
            MOV   A, #Z
            ENDM
            MOV   A, #10
            MOV   A, #20
            MOV   A, #30

```

IRP expansion

CHAPTER 6 PRODUCT UTILIZATION

This chapter introduces some measures recommended for effective utilization of the RA78K4 assembler package.

There are several ways to effectively use the RA78K4 for assembly of source modules. This section introduces just a few of these techniques.

(1) Saving time and trouble in starting up the assembler

Some control instructions have the same functions as assembler options and must always be used when starting up the assembler; examples of these include the processor type specification (-C) and the kanji code specification (-ZS/-ZE/-ZN) (Japanese version only). It is advisable to describe such control instructions in a source module file. In particular, the processor type specification, which cannot be omitted, should be specified in the header section of a source module file using the PROCESSOR control instruction. This avoids the need to specify the assembler option (-C) in the start-up command line each time the assembler program is started. Remember that an error will result if this assembler option is not specified in the start-up command line, in which case the assembler will need to be started from the beginning again with the correct assembler options. The cross-reference list output control instruction (XREF) should also be specified in the module header.

Example

```
        $          PROCESSOR(4038)
        $          DEBUG
        $          XREF

        NAME          TEST

        CSEG
        :
```

(2) How to develop programs with high memory utilization efficiency

The short direct addressing area is an area that can be accessed with instructions of short byte length as compared with other data memory areas.

Therefore, by using this area efficiently, a program with high memory utilization efficiency can be developed.

Declare the short direct addressing area in one module. In this way, even if all the variables intended for location in the short direct addressing area cannot be located there, changes can easily be made so that only variables to be accessed frequently are located in the short direct addressing area.

Module 1

```
        PUBLIC      TMP1, TMP2
WORK    DSEG        AT 0FE20H
TMP1:   DS          2          ;word
TMP2:   DS          1          ;byte
```

Module 2

	EXTRN	TMP1, TMP2
SAB	CSEG	
	MOVW	TMP1, #1234H
	MOV	TMP2, #56H
	:	

APPENDIX A LIST OF RESERVED WORDS

Reserved words are available in six types: machine language instructions, directives, control instructions, operators, register names, and sfr symbols. The reserved words are character strings reserved in advance by the assembler and cannot be used for other than the intended purposes.

Types of reserved words that can be described in the respective fields of a source program are shown below.

Symbol field	No reserved words can be described in this field.
Mnemonic field	Only machine language instructions and directives can be described in this field.
Operand field	Only operators, sfr symbols, and register names can be described in this field.
Comment field	All reserved words can be described in this field.

For the sfr list, refer to the **Special Function Register Table** of each device.

For the interrupt request source list, refer to the **Notes on Use** in each device file.

For the machine language instructions and list of register names, refer to the user's manual of each device.

(1) List of reserved words

Operators	AND	BITPOS	DATAPOS	EQ	GE
	GT	HIGH	HIGHW	LE	LOW
	LOWW	LT	MASK	MOD	NE
	NOT	OR	SHL	SHR	XOR
Directives	AT	BASE	BR	BSEG	CALL
	CALLT0	CSEG	DB	DBIT	DG
	DS	DSEG	DTABLE	DTABLEP	DW
	END	ENDM	EQU	EXITM	EXTBIT
	EXTRN	FIXED	FIXEDA	GRAM	IRP
	LOCAL	LRAM	MACRO	NAME	ORG
	PAGE	PAGE64K	PUBLIC	REPT	RSS
	SADDR	SADDR2	SADDRA	SADDRP	SADDRP2
SET	SFR	SFRP	UNIT	UNITP	
Control instructions	CHGSFR	CHGSFRA	COND	DEBUG	DEBUGA
	EJ	EJECT	ELSE	ELSEIF	_ELSEIF
	ENDIF	FORMFEED	GEN	IC	IF
	_IF	INCLUDE	KANJICODE	LENGTH	LI
	LIST	NOCOND	NODEBUG	NODEBUGA	NOFORMFEED
	NOGEN	NOLI	NOLIST	NOSYMLIST	NOXR
	NOXREF	PC	PROCESSOR	RESET	SET
	ST	SUBTITLE	SYMLIST	TAB	TITLE
TT	WIDTH	XR	XREF		
Others	DGL	DGS	TOL_INF		

APPENDIX B LIST OF DIRECTIVES

(1) List of directives

No.	Directive				Function /Classification	Remarks
	Symbol Field	Mnemonic Field	Operand Field	Comment Field		
1	[segment name]	CSEG	[relocation-attribute]	[:comment]	Declares the start of a code segment.	
2	[segment name]	DSEG	[relocation-attribute]	[:comment]	Declares the start of a data segment.	
3	[segment name]	BSEG	[relocation-attribute]	[:comment]	Declares the start of a bit segment.	
4	[segment name]	ORG	absolute-expression	[:comment]	Declares the start of an absolute segment.	Forward reference of symbols within an operand is prohibited.
5	name	EQU	expression	[:comment]	Defines a name.	name: symbol Forward or external reference of symbols within an operand is prohibited.
6	name	SET	absolute-expression	[:comment]	Defines a redefinable name.	name: symbol Forward reference of symbols within an operand is prohibited.
7	[label:]	DB	{{(size) initial value [,...]}}	[:comment]	Initializes or reserves a byte data area.	label: symbol A character string can be located in place of an initial value.
8	[label:]	DW	{{(size) initial value [,...]}}	[:comment]	Initializes or reserves a word data area.	label: symbol
9	[label:]	DG	{{(size) initial value [,...]}}	[:comment]	Initializes or reserves a 3-byte data area.	label: symbol
10	[label:]	DS	absolute-expression	[:comment]	Reserves byte data area.	name: symbol Forward reference of symbols within an operand is prohibited.
11	name	DBIT	None	[:comment]	Reserves a bit data area.	name: symbol Forward reference of symbols within an operand is prohibited.
12	[label:]	PUBLIC	symbol-name [,...]	[:comment]	Declares an external definition name.	
13	[label:]	EXTRN	symbol-name [,...]	[:comment]	Declares an external reference name.	
14	[label:]	EXTBIT	bit-symbol-name [,...]	[:comment]	Declares an external reference name.	Symbol names are limited to those having a bit value.
15	[label:]	NAME	object-module-name	[:comment]	Defines a module name.	module name: symbol
16	[label:]	BR	expression	[:comment]	Automatically selects a branch instruction.	label: symbol

APPENDIX B LIST OF DIRECTIVES

No.	Directive				Function /Classification	Remarks
	Symbol Field	Mnemonic Field	Operand Field	Comment Field		
17	[label:]	CALL	expression	[:comment]	Automatically selects the CALL instruction.	label: symbol
18	[label:]	RSS	n	[:comment]	Declares the value of the register set selection flag.	n = 0, 1
19	macro-name	MACRO	[formal-parameter [...]]	[:comment]	Defines a macro.	macro-name: symbol
20	[label:]	LOCAL	symbol-name [...]	[:comment]	Defines a symbol valid only within a macro.	Can only be used in the macro definition.
21	[label:]	REPT	absolute-expression	[:comment]	Specifies repeat count during macro expansion.	label: symbol
22	[label:]	IRP	formal-parameter, <actual-parameter [...]>	[:comment]	Assigns an actual parameter to a formal parameter.	label: symbol
23	[label:]	EXITM	None	[:comment]	Interrupts macro expansion.	Can only be used in the macro definition.
24	None	ENDM	None	[:comment]	Terminates macro definition.	Can only be used in the macro definition.
25	None	END	None	[:comment]	Indicates the end of the source module.	

APPENDIX C MAXIMUM PERFORMANCE CHARACTERISTICS

(1) Maximum performance characteristics of assembler

Item	Maximum Performance Characteristics	
	PC Version	WS Version
Number of symbols (local + public)	65,535 symbols ^{Note 1}	65,535 symbols ^{Note 1}
Number of symbols for which cross-reference list can be output	65,534 symbols ^{Note 2}	65,534 symbols ^{Note 2}
Maximum size of macro body for one macro reference	1 MB	1 MB
Total size of all macro bodies	10 MB	10 MB
Number of segments in one file	256 segments	256 segments
Macro and include specifications in one file	10,000	10,000
Macro and include specifications in one include file	10,000	10,000
Relocation data ^{Note 3}	65,535 items	65,535 items
Line number data	65,535 items	65,535 items
Number of BR directives in one file	32,767 directives	32,767 directives
Number of characters per line	2,048 characters ^{Note 4}	2,048 characters ^{Note 4}
Symbol length	256 characters	256 characters
Number of definitions of switch name ^{Note 5}	1,000	1,000
Character length of switch name ^{Note 5}	31 characters	31 characters
Number of nesting levels on include file in one file	8 levels	8 levels

Notes 1. XMS is used. If there is no XMS, a file is used.

2. Memory is used. If there is no memory, a file is used.

3. "Relocation data" is the data transferred to the linker when the assembler cannot determine the symbol values.

For example, when referring to an external reference symbol by a MOV instruction, two items of relocation data are generated in the .rel file.

4. This includes the carriage return and feed codes. If 2,049 characters or more are described on a line, a warning message is output and any characters at or over 2,049 are ignored.

5. Switch name is set to true or false by SET/RESET directives and used with \$IF, etc.

(2) Maximum performance characteristics of linker

Item	Maximum Performance Characteristics	
	PC Version	WS Version
Number of symbols (local + public)	65,535 symbols	65,535 symbols
Line number data of same segment	65,535 items	65,535 items
Number of segments	65,535 segments	65,535 segments
Number of input modules	1,024 modules	1,024 modules

APPENDIX D INDEX

??RAn	137
?BSEG	34, 94
?BSEGG	34, 94
?BSEGS	34, 94
?BSEGS2	34, 94
?BSEGSA	34, 94
?BSEGSP	34, 94
?BSEGSP2	34, 94
?BSEGUP	34, 94
?CSEG	34, 85
?CSEGB	34, 85
?CSEGFx	34, 85
?CSEGFxA	34, 85
?CSEGT0	34, 85
?CSEGP	34, 85
?CSEGP64	34, 85
?CSEGUP	85
?CSEG	34, 89
?DSEGDT	34, 89
?DSEGDTp	34, 89
?DSEGG	34, 89
?DSEGP	34, 89
?DSEGP64	34, 89
?DSEGS	34, 89
?DSEGSP	34, 89
?DSEGSP2	34, 89
?DSEGUP	89

[A]

Absolute assembler	16
Absolute segment	23, 81, 96
Absolute term	61, 78
Actual parameter	195, 212
ADDRESS	35, 78
ADDRESS term	65
Alphabetic character	30
AND operator	44, 49
Area reservation directive	106
Assembler	13, 20
Assembler option	22, 152
Assembler package	13
Assembly language	14
Assembly list control instruction	164
Assembly termination directive	149
AT relocation attribute	84, 85, 88, 92

Automatic branch instruction selection directive	125
---	-----

[B]

BASE relocation attribute	84, 85
Backward reference	75
Binary number	37
BIT	35
Bit access	67
Bit segment	28, 81, 91
Bit symbol	69
BITPOS operator	44, 58
BR directive	20, 126
BSEG directive	91

[C]

CALL directive	128
CALLF instruction	84
CALLT0 relocation attribute	84, 85
CALLT instruction	84
Character set	29
Character-string constant	38
CHGSFR control instruction	22, 191
Code segment	23, 81, 83
Comment field	42, 204
Concatenation	200
COND control instruction	171
Conditional assembly function ..	20, 144, 171, 183
Constant	37
Control instruction	22, 151
Cross-reference list output specification control instruction	157
CSEG directive	83

[D]

Data segment	23, 81, 87
DATAPOS operator	44, 58
DB directive	107
DBIT directive	115
DEBUG control instruction	22, 155
Debug information output control instruction	154
DEBUGA control instruction	22, 156
Decimal numbers	37

DG directive	111
Directives	80, 205
DS directive	113
DSEG directive	87
DTABLE relocation attribute	88, 89
DW directive	109

[E]

EJECT control instruction	165
ELSE control instruction	184
ELSEIF control instruction	184
END directive	150
ENDIF control instruction	184
ENDM directive	154
EQ operator	44, 51
EQU directive	100
EXITM directive	144
Expressions	44
EXTBIT directive	119
External definition declaration	116, 121
External reference declaration	116, 117, 119
External reference term	61, 78
EXTRN directive	117

[F]

FIXED relocation attribute	84, 85
FIXEDA relocation attribute	84, 85
Formal parameter	135, 193, 200
FORMFEED control instruction	22, 179
Forward reference	75

[G]

GE operator	44, 52
GEN control instruction	169
General-purpose register	39
General-purpose register pair	39
General-purpose register selection directive	39
Global symbol	137, 197
GT operator	44, 52

[H]

Hexadecimal number	37
HIGH operator	44, 56
HIGHW operator	44, 57

[I]

<i>idea-L</i> editor	13
IF control instruction	184
INCLUDE control instruction	161
Inclusion control instruction	160
IRP directive	142
IRP-ENDM block	142

[L]

Label	32
LE operator	44, 53
LENGTH control instruction	22, 181
Librarian	13
Lines	28
Linkage directive	116
Linker	13, 18
LIST control instruction	167
List converter	13
LOCAL directive	137
LOCAL symbol	197
LOW operator	44, 56
LOWW operator	44, 57
LT operator	44, 53

[M]

Machine language	14
Macros	20, 193
Macro body	135, 137, 144, 195
Macro definition	169, 144
MACRO directive	135, 193
Macro directive	134
Macro expansion	169, 196
Macro name	32, 135, 194, 195
Macro operator	200
Macro reference	169, 195
MASK operator	44, 59
Memory initialization directive	106
Mnemonic	36
Mnemonic field	36, 204
MOD operator	44, 48
Modular programming	16
Module body	21, 23
Module header	21, 22
Module name	32, 123, 124
Module tail	21, 24

[N]

Name 32, 100
 NAME directive 124
 NE operator..... 44, 51
 NOCOND control instruction 171
 NODEBUG control instruction 22, 155
 NODEBUGA control instruction 22, 156
 NOFORMFEED control instruction 22, 179
 NOGEN control instruction..... 169
 NOLIST control instruction 167
 NOSYMLIST control instruction 22, 159
 NOT operator 44, 49
 NOXREF control instruction 22, 158
 NUMBER..... 35, 78
 Number of files..... 18
 NUMBER term 65
 Numeric character..... 29
 Numeric constant 37

[O]

Object converter..... 14
 Object module 123, 154
 Octal number 37
 Operand 37, 70, 72
 Operand field 37, 204
 Operator 44
 Order of precedence of operator..... 45
 Optimize function 20
 OR operator 44, 50
 ORG directive 96

[P]

PAGE relocation attribute 84, 85, 88, 89
 PAGE64K relocation attribute 84, 85, 88, 89
 PROCESSOR control instruction 22, 153
 Processor type specification control
 instruction..... 152
 Project Manager..... 13
 PUBLIC directive..... 121

[R]

Register set selection flag..... 131
 Relocatable assembler 16
 Relocatable term 61, 78
 Relocation attribute 61, 75
 REPT directive 140
 REPT-ENDM block 140

RESET control instruction 188
 RSS flag 131
 RSS directive..... 131

[S]

SADDR relocation attribute 88, 89, 92, 94
 SADDR2 relocation attribute 88, 89, 92, 94
 SADDRA relocation attribute 88, 89, 92, 94
 SADDRP relocation attribute 88, 89, 94
 SADDRP2 relocation attribute 88, 89, 94
 Segment 18, 23, 81
 Segment definition directive 81
 Segment name 85, 89, 94, 97
 SET control instruction 188
 SET directive 104
 SHL operator 44, 55
 SHR operator..... 44, 54
 Source module 21, 152
 Special character..... 30, 40
 Special function register 39
 Statement 28
 Structured assembler preprocessor 13
 Subroutine 193
 SUBTITLE control instruction 176
 Subtitle section 176
 Switch name 185, 188
 Symbol..... 18, 32, 197, 214
 Symbol attribute..... 34, 75
 Symbol definition directive..... 99
 SYMLIST control instruction 22, 159

[T]

TAB control instruction 22, 182
 TITLE control instruction..... 22, 173

[U]

UNIT relocation attribute..... 84, 85, 88, 89, 92, 94
 UNITP relocation attribute 84, 85, 88, 89, 94

[W]

WIDTH control instruction..... 22, 180

[X]

XOR operator 44, 50
 XREF control instruction..... 22, 158

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.

<p>North America NEC Electronics Inc. Corporate Communications Dept. Fax: +1-800-729-9288 +1-408-588-6130</p>	<p>Hong Kong, Philippines, Oceania NEC Electronics Hong Kong Ltd. Fax: +852-2886-9022/9044</p>	<p>Asian Nations except Philippines NEC Electronics Singapore Pte. Ltd. Fax: +65-250-3583</p>
<p>Europe NEC Electronics (Europe) GmbH Technical Documentation Dept. Fax: +49-211-6503-274</p>	<p>Korea NEC Electronics Hong Kong Ltd. Seoul Branch Fax: +82-2-528-4411</p>	<p>Japan NEC Semiconductor Technical Hotline Fax: +81- 44-435-9608</p>
<p>South America NEC do Brasil S.A. Fax: +55-11-6462-6829</p>	<p>Taiwan NEC Electronics Taiwan Ltd. Fax: +886-2-2719-5951</p>	

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	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Technical Accuracy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>