To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

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

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

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

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

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

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

User’s Manual

Phase-out/Discontinued

RA75X ASSEMBLER PACKAGE

VERSION 5.XX

Language

Document No.   U12385EJ7V0UM00
(Previous No. EEU-1363D)
Date Published July 1997 N

[MEMO]

MS-DOS is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.
PC/AT and PC DOS are trademarks of International Business Machines Corporation.

The information in this document is subject to change without notice.
No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Corporation. NEC Corporation assumes no responsibility for any errors which may appear in this document.
NEC Corporation does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from use of a device described herein or any other liability arising from use of such device. No license, either express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Corporation or of others.

M7A 96.10

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: 800-366-9782
Fax: 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.1.
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: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

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.
United Square, Singapore 1130
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-719-2377
Fax: 02-719-5951

NEC do Brasil S.A.
Sao Paulo-SP, Brasil
Tel: 011-889-1680
Fax: 011-889-1689

J96. 8

Major Revisions in This Version

Section	Description
Whole manual	RA75X Assembler Package Version 4.5X → Version 5.XX
Whole manual	Change of each program of the RA75X Assembler Package • With addition of macro function to assembler program, deletion macro processor • Addition of library converter program
Whole manual	Addition of target devices: μPD750064, 750066, 750068, 75P0076, 750104, 750106, 750108, 75P0116, 753012A, 753016A, 753017A, 75P3018A, 753036, 75P3036, 753204, 753206, 753208, 75P3216, 753304[Note], 754202, 754144, 754244, 754264, 75F4264[Note], 754302, 754304, 75P4308
Whole manual	Change: Target device under development → development completed: μPD750004, 750006, 750008, 75P0016, 753012, 753016, 753017, 75P3018, 753104, 753106, 753108, 75P3116
p.25, p.88, p.89	Change: Symbol length: 1 to 8 characters → 1 to 31 characters (1 to 8 characters when -NS option is specified)
p.35	Change of a part of **Table 3-8. Operator Priority Order**
p.110	Addition to Caution in **4.6 BRANCH INSTRUCTION AUTO SELECT PSEUDO-INSTRUCTIONS (1) BR**
p.121 to p.140	Addition of **CHAPTER 5 MACRO**
p.151	Addition of **6.4 CONDITIONAL ASSEMBLE CONTROL INSTRUCTIONS**
p.174 to p.186	Addition of control instruction and Note to **APPENDIX B LIST OF RESERVED WORDS**
p.190	Addition of (5) Other to **APPENDIX D LIST OF MAXIMUM PERFORMANCE CAPABILITIES**

The mark ★ shows major revised points.

Note Under development

[MEMO]

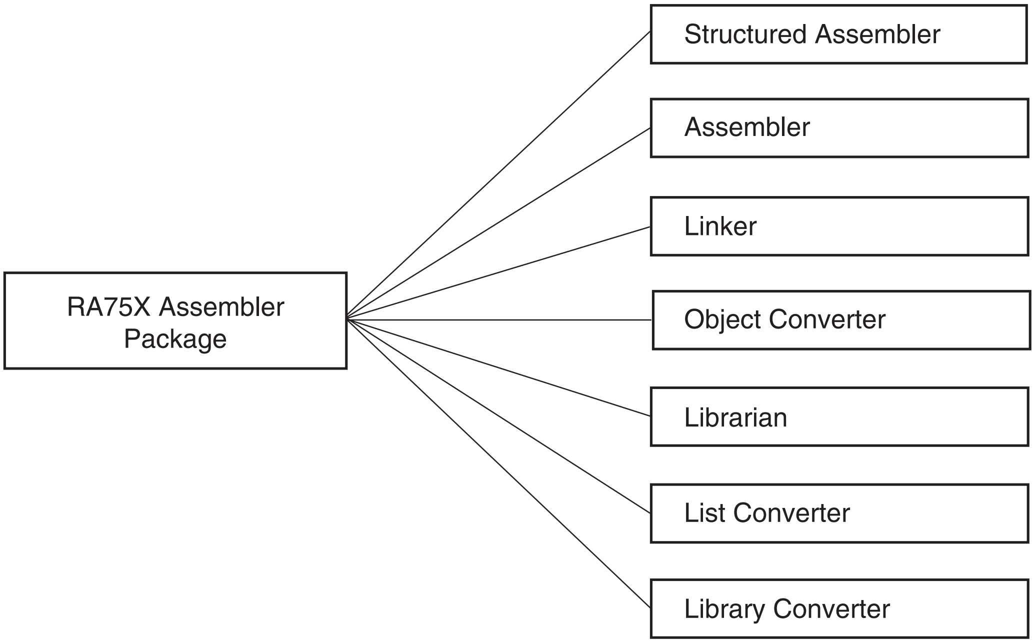
PREFACE

This manual has been prepared so that the basic functions of each program of the RA75X assembler package (subsequently referred to as “assembler package”) and the source program describing procedure can be understood correctly.

This manual does not describe the operating procedure for each program. Therefore, after reading this manual, be sure to read the **RA75X Assembler Package User’s Manual Operation (U12622E)** when operating each program. (Subsequently referred to as **Operation**)

This manual applies to assembler package products of version 5.XX.

★



[Intended Readership]

This manual is intended for use by those who have an understanding of the microcontroller (75X Series/75XL Series) functions and instructions to be developed.

[Target Devices]

The following microcontroller's software can be developed using this assembler package.

<75X Series>

Series	Title	Target Device
—	Evachip	μPD75000, 75000A
General-purpose series	General-purpose	μPD75004, 75006, 75008, 75P008
	General-purpose + A/D converter	μPD75028, 75036, 75P036, 75064, μPD75066, 75068, 75P068
	General-purpose + A/D converter + EEPROM	μPD75048, 75P048
Control series	For control	μPD75104, 75106, 75108, 75112, μPD75116, 75104A, 75108A, μPD75P108, 75P108B, 75P116
	For low-voltage high-speed control	μPD75108F, 75112F, 75116F
	F products + low voltage	μPD75116H, 75117H, 75P117H
FIP drive series	For FIP drive	μPD75206, 75208, 75212A, μPD75216A, 75217, 75218, μPD75P216A, 75P218, 75268, μPD75CG208, 75CG216A
	FIP drive + A/D converter	μPD75236, 75237, 75238, 75P238
LCD drive series	For LCD drive	μPD75304, 75306, 75308, 75304B, μPD75306B, 75308B, 75312, 75316, μPD75312B, 75316B, 75P308, μPD75P316, 75P316A, 75P316B
	LCD drive + A/D converter	μPD75328, 75P328
	LCD drive + A/D converter + advanced function	μPD75336, 75P336
Slave series		μPD75402A, 75P402
Control (A/D converter on-chip) series	For control (A/D converter on-chip)	μPD75512, 75516, 75P516
	For control (A/D converter on-chip) + high speed	μPD75517, 75518, 75P518
Telephone series	LCD drive + DTMF + D/A converter	μPD75352A
	LCD drive + DTMF + D/A converter + A/D converter	μPD75617A

★ <75XL Series>

Series	Title	Target Device
General-purpose series	General-purpose	μPD750004, 750006, μPD750008, 75P0016
	General-purpose + RC oscillator	μPD750104, 750106, μPD750108, 75P0116
	General-purpose + A/D converter	μPD750064, 750066, μPD750068, 75P0076
Series for LCD drive	For LCD drive	μPD753012, 753012A, 753016, μPD753016A, 753017, 753017A, μPD75P3018, 75P3018A
	For LCD drive + A/D converter	μPD753036, 75P3036
	For LCD drive (small)	μPD753104, 753106, μPD753108, 75P3116
		μPD753204, 753206, μPD753208, 75P3216
	For LCD drive + RC oscillator (small)	μPD753304[Note]
Key-less entry series		μPD754202
		μPD754144, 754244, μPD754264, 75F4264[Note]
General-purpose small-size series		μPD754302, 754304, 75P4308

Note Under development

Caution A device file, which must be purchased separately, is required for the development of a 75XL Series device.

[Format]

This manual consists of the following chapters.

CHAPTER 1 GENERAL DESCRIPTION CHAPTER 2 75X SERIES/75XL SERIES FEATURES

These two chapters outline the functions of the entire assembler package including the assembler package roles for microcontroller development.

CHAPTER 3 SOURCE PROGRAM DESCRIPTION METHOD

This chapter describes source program description rules including the configuration of the source program, description grammars, and assembler operators.

CHAPTER 4 PSEUDO-INSTRUCTIONS CHAPTER 5 MACRO CHAPTER 6 CONTROL INSTRUCTIONS

These chapters deal with assembler pseudo-instructions, macros, and control instructions using examples concerning the procedure for writing and using those instructions.

CHAPTER 7 ASSEMBLER PACKAGE UTILIZATION

This chapter introduces know-how concerning source program description.

APPENDIX

The appendixes list assemble objective devices, reserved words, pseudo-instructions, maximum performance capabilities, precautions, and index.

This manual makes no detailed description of instructions. For details of the instructions, refer to the user's manual of each target device to be developed.

[Reading the Manual]

Those who use the assembler for the first time should start with **CHAPTER 1 GENERAL DESCRIPTION**. Those who have general knowledge of the assembler can skip **CHAPTER 1 GENERAL DESCRIPTION**.

There are several rules relating to the 75X Series/75XL Series source program description procedure. Carefully read **CHAPTER 3 SOURCE PROGRAM DESCRIPTION METHOD**.

Those who want to know the pseudo-instructions, macros, and control instructions of the assembler should read **CHAPTERS 4**, **5**, and **6**. These three chapters describe instruction formats, functions and applications.

[Legend]

The symbols have the following meanings in this manual.

... : The same format is repeated.

[ ] : Values inside brackets can be omitted.

“ ” : Character(s) or character string(s) marked by “ ”

‘ ’ : Character(s) marked by ‘ ’

( ) : Character(s) marked by ( )

< > : Character(s), the title in particular, marked by < >

“ ” : Character(s) marked by “ ”

__ : Input character string(s) or important portion(s)

⎵ : One or more blank space

△ : One blank space

⋮ : Abbreviated program description

CR : Carriage return

LF : Line feed

/ : Demarcation symbol

● - ▲ : From ● to ▲

[Related Documents]

The following are documents related to this manual.

	Document Name	Document No.	
		English	Japanese
★	RA75X Assembler Package Version 5.XX User’s Manual <Operation>	Under planning	U12622J
★	RA75X Structured Assembler Preprocessor User’s Manual	Under planning	U12598J
	75X Series Structured Assembler Preprocessor Application Note	EEA-1203	EEA-603

[MEMO]

CONTENTS

CONTENTS OF FIGURES

[MEMO]

CONTENTS OF TABLES

Table No. Title Page

[MEMO]

CHAPTER 1 GENERAL DESCRIPTION

In this chapter, the assembler package role in 75X Series/75XL Series development will be described.

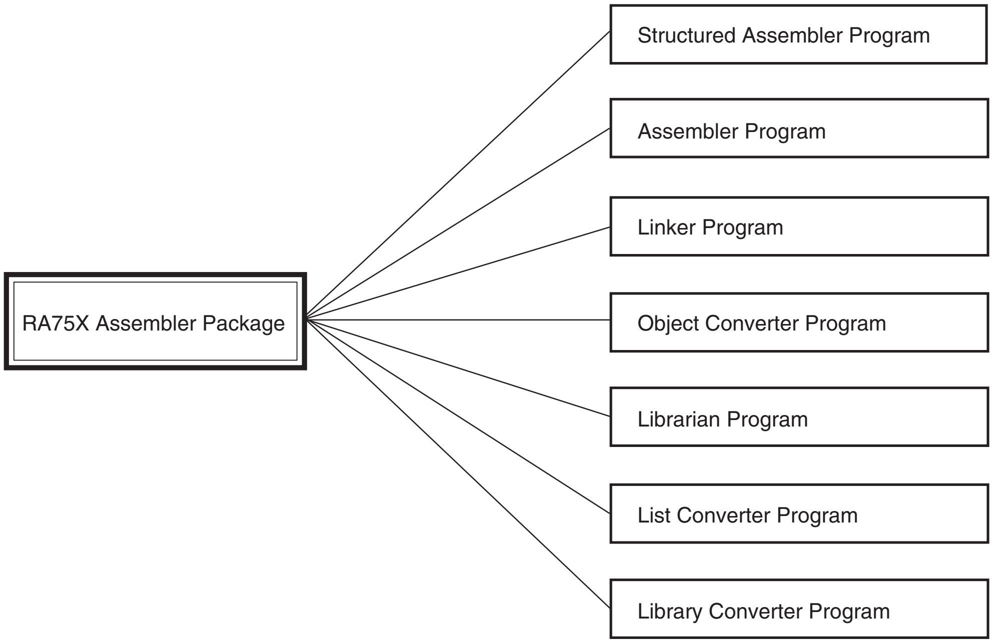
1.1 OUTLINE OF ASSEMBLER

The RA75X assembler package (subsequently referred to as assembler package) is a general term for a seriesof programs used to convert the source program described by 75X Series/75XL Series assembler language to machine codes.

This assembler package consists of seven programs; a structured assembler, an assembler, a linker, an object converter, a librarian, a list converter, and a library converter.

Figure 1-1. RA75X Assembler Package

★



1.1.1 Assembler

(1) Assembly language and machine code

The assembly language is the most basic programming language for microcomputers.

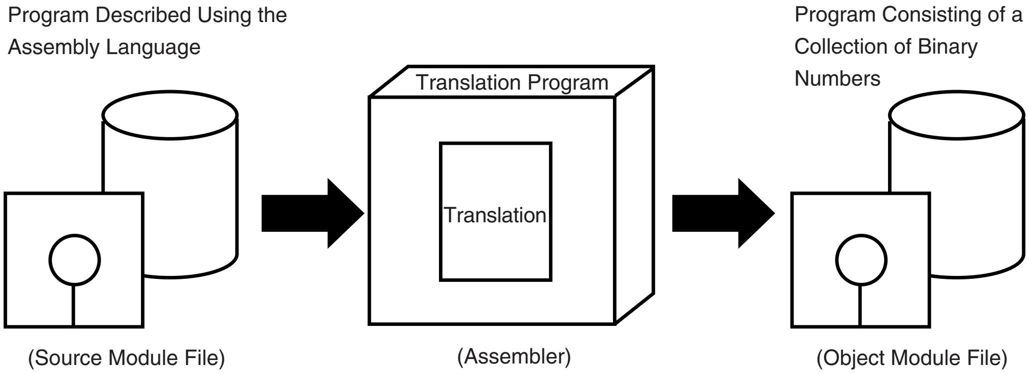
Programs and data are necessary for microcomputer operations. The human operator will carry out programming to store them into the microcomputer memory unit. The programs and data that the microcomputer can handle are a collection of binary numbers called machine codes (the words which the computer can understand).

We may have difficulty or make errors in creatingprograms using the machine codes, that is binary numbers.

Thus, the meaning of the machine codes is represented by easily understood English symbolic codes and the symbolic codes in turn are used for program creation. The program language system based on those codes is called an assembly language.

An assembler is a program to translate the program created using the assembler language into a collection of binary numbers which the microcomputer can understand.

Figure 1-2. Assembler Flow



(2) Development of microcomputer applied products and the roles of assembler package

Figure 1-3. Development Process of Microcomputer Applied Products shows where programming with the assembly language is positioned in product development.

Figure 1-3. Development Process of Microcomputer Applied Products

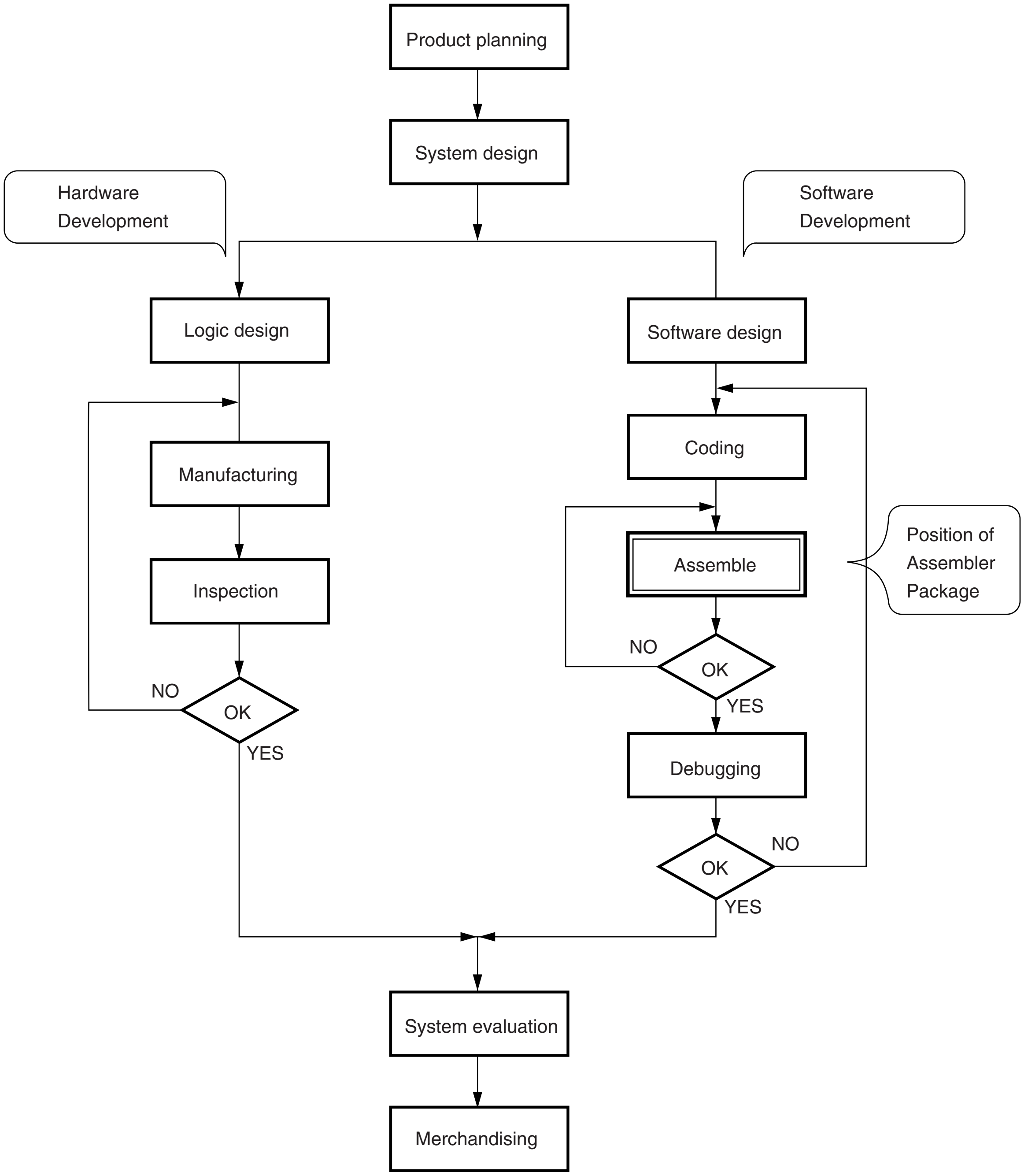
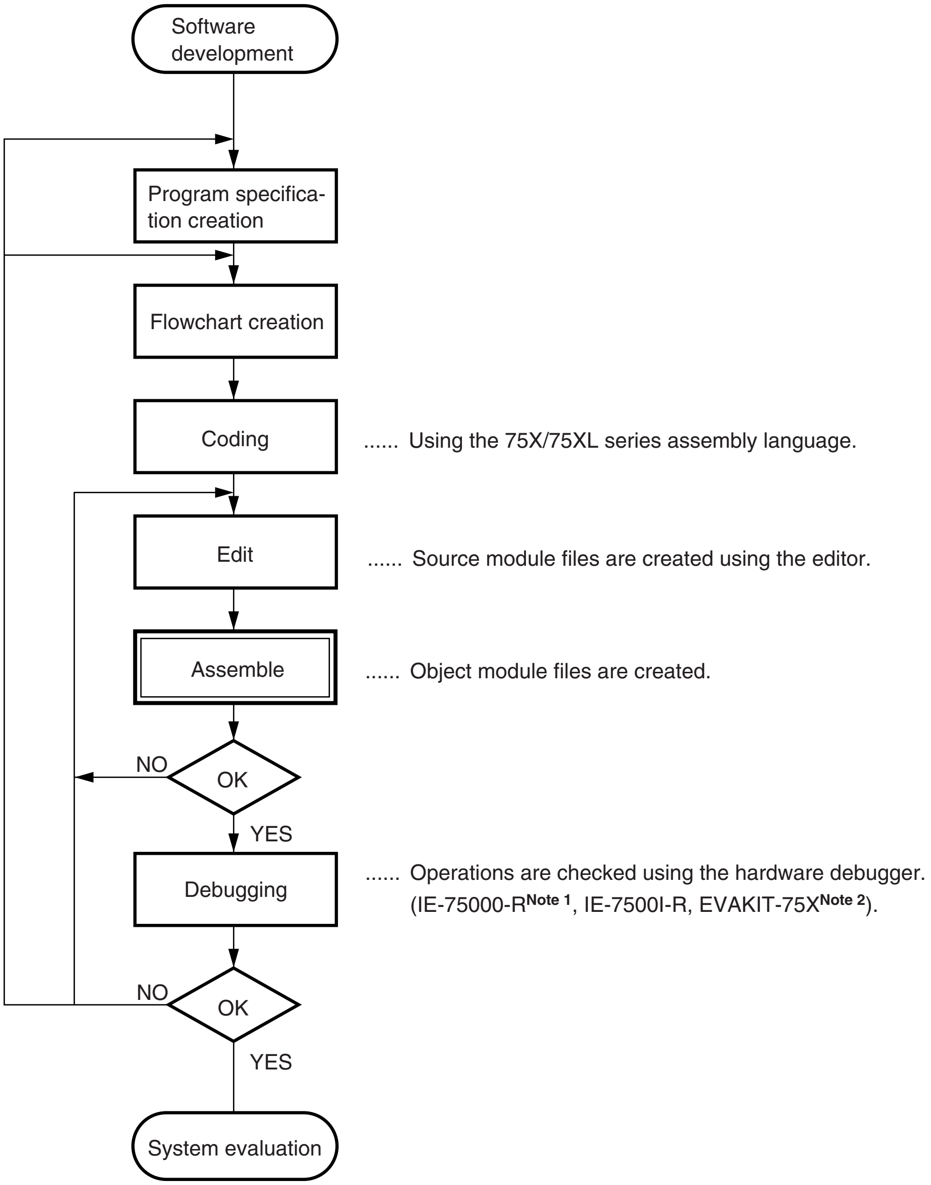


Figure 1-4. Software Development Process shows the software development process in more detail.

Figure 1-4. Software Development Process

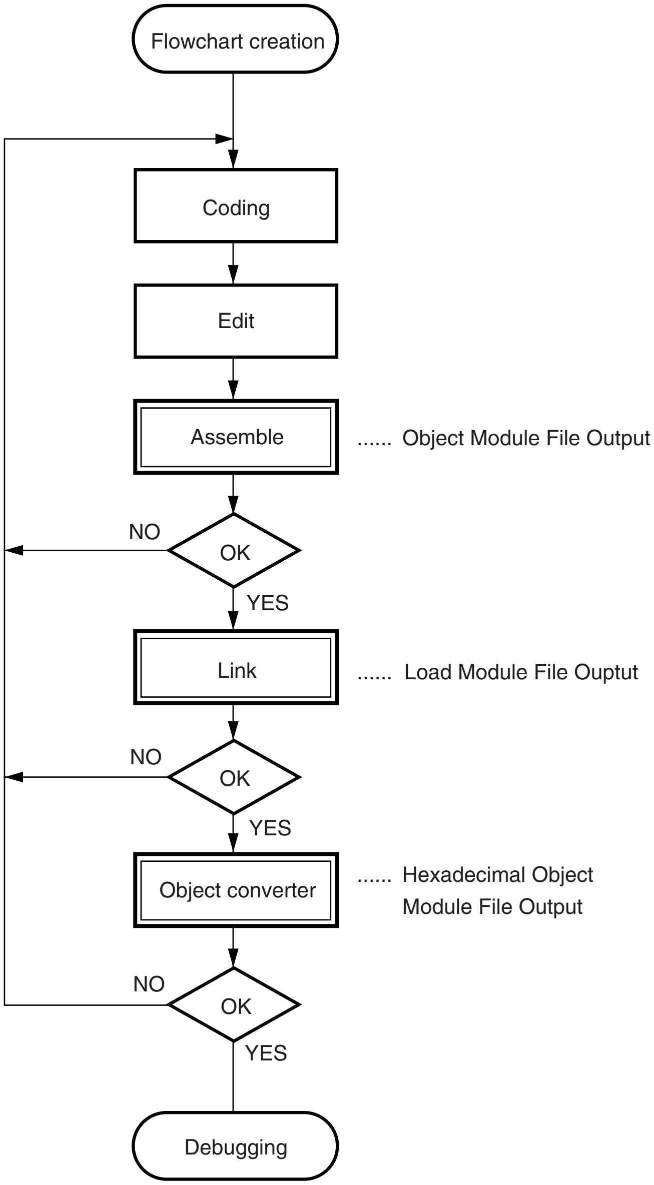

Software development
Program specification creation
Flowchart creation
Coding
...... Using the 75X/75XL series assembly language.
Edit
...... Source module files are created using the editor.
Assemble
...... Object module files are created.
NO
OK
YES
Debugging
...... Operations are checked using the hardware debugger. (IE-75000-R[Note 1], IE-7500I-R, EVAKIT-75X[Note 2]).
NO
OK
YES
System evaluation

Notes 1. Maintenance product (No longer available for purchase)

2. Discontinuation product (No longer available for purchase)

The assembler package is now inserted in the assembling process.

Figure 1-5. Assembling Process for Assembler Package


Flowchart creation
Coding
Edit
Assemble
...... Object Module File Output
NO
OK
YES
Link
...... Load Module File Ouptut
NO
OK
YES
Object converter
...... Hexadecimal Object Module File Output
NO
OK
YES
Debugging

★ This package has the following features.

1. **Branch instruction optimization function**
   A BR pseudo-instruction is provided. This directive automatically selects an appropriate branch instruction code format.
   Conventionally, for branch operations, it is required to select either 2-byte or 1-byte branch instructions according to the branch instructions' destination range because it is critical for the efficiently use of memory resources.
   However, it is a lot of work for a programmer to take the destination range into account every time a branch instruction is described.
   To avoid this, use the BR directive, which makes the assembler generate appropriate branch instruction codes according to the destination range. This feature is called branch instruction optimization function.

2. **VENTn pseudo-instruction**
   This pseudo-instruction facilitates writing to the vector table. The 75X Series/75XL Series devices have an interrupt vector table at addresses 0000H to 000FH (The size depends on the part number). This vector table can hold the starting address of interrupt services, the setting of the memory bank enable flag (MBE), and the value of register bank enable flag (RBE) during an interrupt servicing.

3. **TCALL, TBR pseudo-instructions**
   These pseudo-instructions facilitate setting data to the GETI instruction reference table.
   When 2-byte or 3-byte branch instructions or call instructions need to be executed as 1-byte instructions, special data must be set in the reference table (0020H to 007FH). This setting work can be facilitated by the use of the GETI instruction.

4. **Librarian (LB75X)**
   The library function integrates plural object modules into a library file.
   Integrating general-purpose modules into a single file improves the module use efficiency. It also contributes to improved file management and operation efficiency.

5. **List converter (LCNV75X)**
   The list converter improves the debugging work efficiency when a program assembled by a relocatable assembler is debugged with the IE-75000-R**Note 1**, IE-75001-R, or EVAKIT-75X**Note 2**.
   Usually, the assemble list values eventually do not match the object codes which reference the addresses and relocatable symbols in the relocatable segments. For this reason, if absolute addresses need to be specified for debugging, it is required to refer to the link map list because the assemble list alone cannot show the absolute addresses.
   The list converter is the program that eliminates the reference requirement. This program replaces the relocatable addresses and object codes in the assemble list, which is output from the assembler, with the eventually determined absolute addresses, to generate an absolute assemble list.

   Notes 1. Maintenance product (No longer available for purchase)
   2. Discontinuation product (No longer available for purchase)

6. **Macro**
   A macro is a labeled series of instructions. Only the label must be written in the source program in place of the corresponding instructions.
   If there are instruction groups that are frequently used, use of macros is effective because it lightens the source program. Also, if a function is composed of specific instruction series, labeling these series instructions as a macro makes the program simple and easy to code or revise.

1.1.2 Relocatable Assembler

The machine code converted by the assembler is written into the microcomputer memory for use. Before the writing operation, the location where the converted machine code should be written in the memory must be determined.

Thus, the machine code to be converted with the assembler is provided with the information concerning **"At which address in the memory each machine code should be positioned"**.

Depending on the method of positioning the machine codes at the memory addresses, the assemblers are roughly classified into **"absolute assemblers"** and **"relocatable assemblers"**.

- **Absolute assembler**

  The machine codes converted by one assembly are positioned at the absolute addresses.

- **Relocatable assembler**

  The addresses of the machine codes converted by one assembly are temporary addresses. Absolute addresses are determined using a program called 'linker'.

When creating one program using an absolute assembler, the program must have been created by programming at one time in principle. However, if a large program is created at one time, it may become complex and program analysis for maintenance may become difficult. Thus, one program is divided into several subprograms (modules) for each function unit. This program development process is called 'modular programming'.

The relocatable assembler is suitable for modular programming.

Modular programming using the relocatable assembler makes it possible to obtain the following advantages.

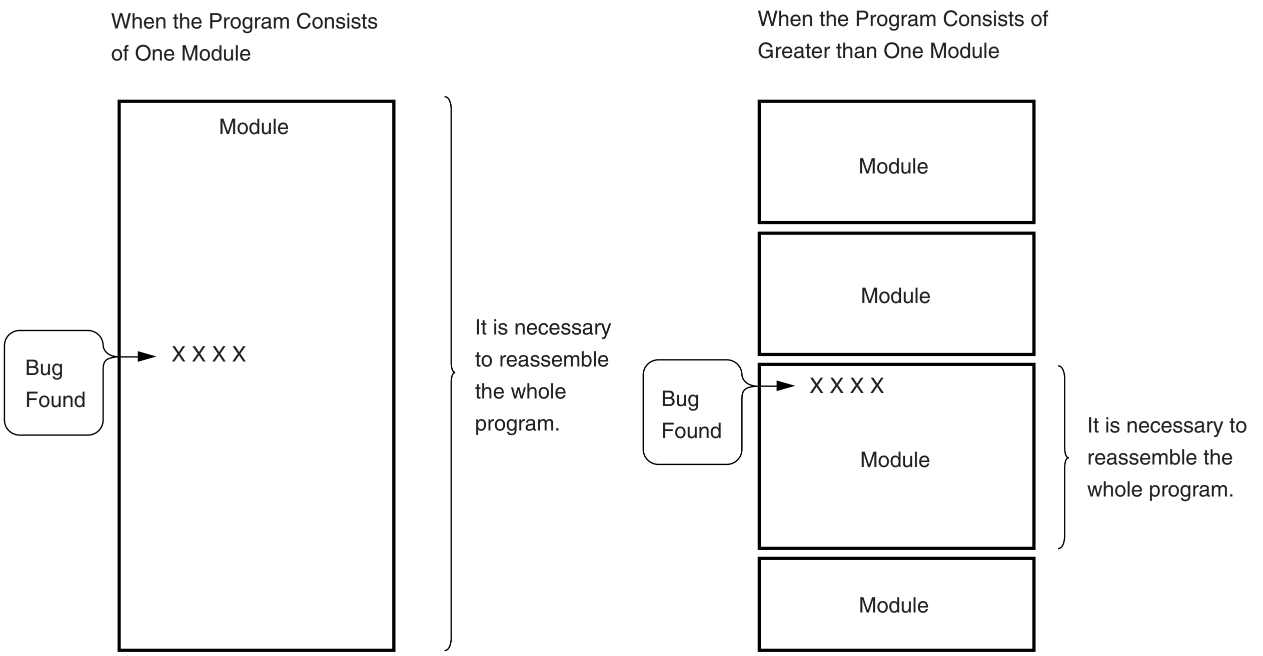
(1) Improvement of development efficiency

It is difficult to execute programming for a large program at one time. In such cases, dividing the program into function modules enables several people to develop the program concurrently with improved efficiency.

If a bug is found in the program, it is not necessary to assemble the whole program for partial correction. It is possible to reassemble only the modules requiring correction.

By so doing, the debugging time can be decreased.

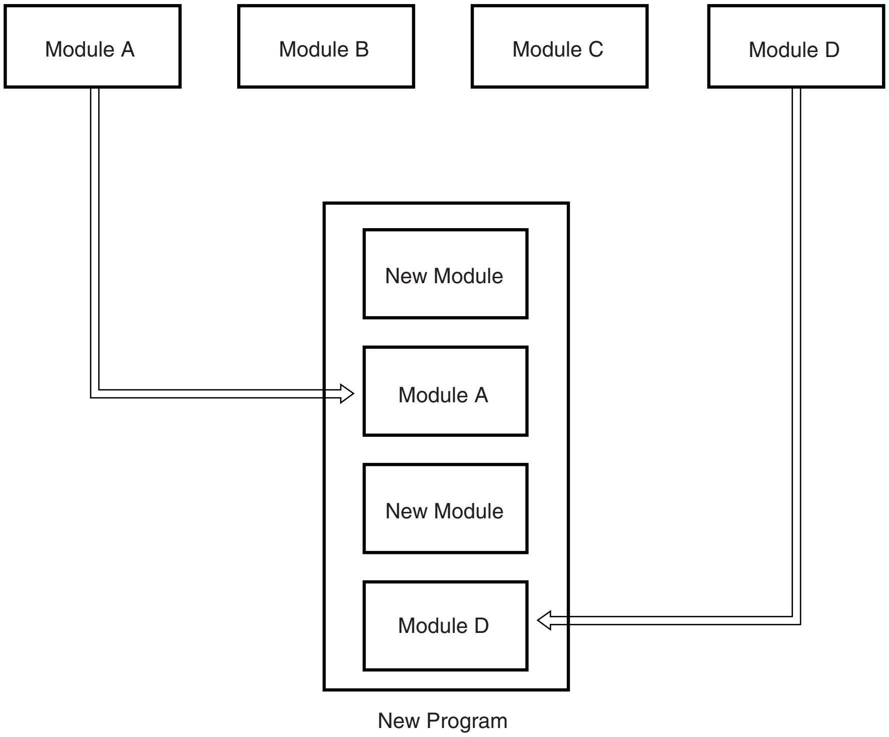
Figure 1-6. Renewing Assembly


When the Program Consists of One Module
Module
Bug Found
X X X X
It is necessary to reassemble the whole program.
When the Program Consists of Greater than One Module
Module
Module
Bug Found
X X X X
Module
It is necessary to reassemble the whole program.
Module

(2) Utilization of resources

Previously generated highly-reliable, highly-universal modules can be utilized for the development of another program. By accumulating those highly-universal modules, the extent of new program development can be decreased.

Figure 1-7. Program Creation Using Existing Modules



[MEMO]

CHAPTER 2 75X SERIES/75XL SERIES FEATURES

In this chapter, features of the memory space of the 75X Series/75XL Series which is the target of the assembler package will be described.

2.1 MEMORY FEATURES

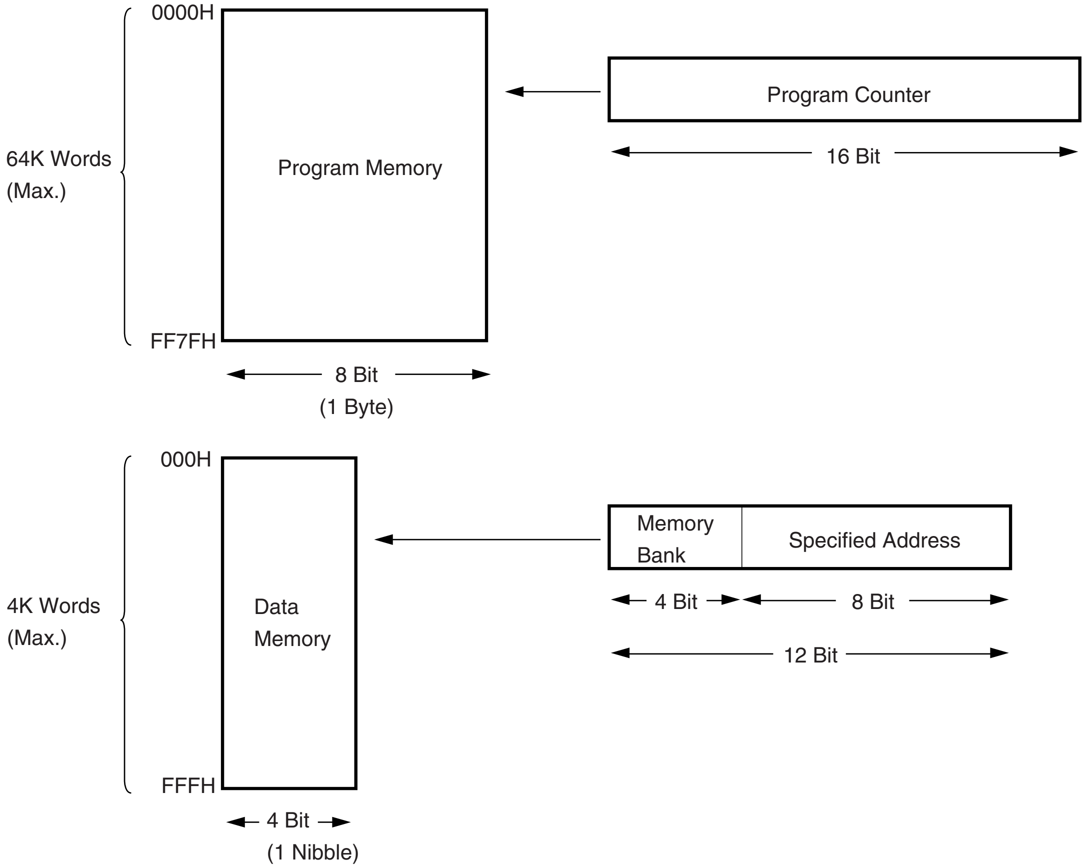
The 75X Series has the following maximum memory spaces:

Program memory (ROM)	:	64K words × 8 bits (64 Kbytes)
Data memory (RAM)	:	4K words × 4 bits (4K nibbles)

The program memory and data memory spaces are separate. The program memory has a 1-word, 8-bit configuration and the data memory has a 1-word, 4-bit configuration.

As shown in **Figure 2-1**, the program memory is addressed by a 16-bit program counter and the data memory is addressed by a total of 12 bits consisting of 4 bits of the memory bank (MB) and 8 bits of the address directly or indirectly specified by an instruction.

Figure 2-1. Memory Addressing



2.2 MEMORY AND SEGMENT DEFINITION PSEUDO-INSTRUCTIONS

The relocatable assembler is provided with the segment definition pseudo-instructions to define the memory area to cope with the memory configuration described in section **2.1 MEMORY FEATURES**.

CSEG pseudo-instruction and DSEG pseudo-instruction are segment definition pseudo-instructions.

The CSEG pseudo-instruction is used to define the use of program memory area and the DSEG pseudo-instruction is used to define the use of data memory area.

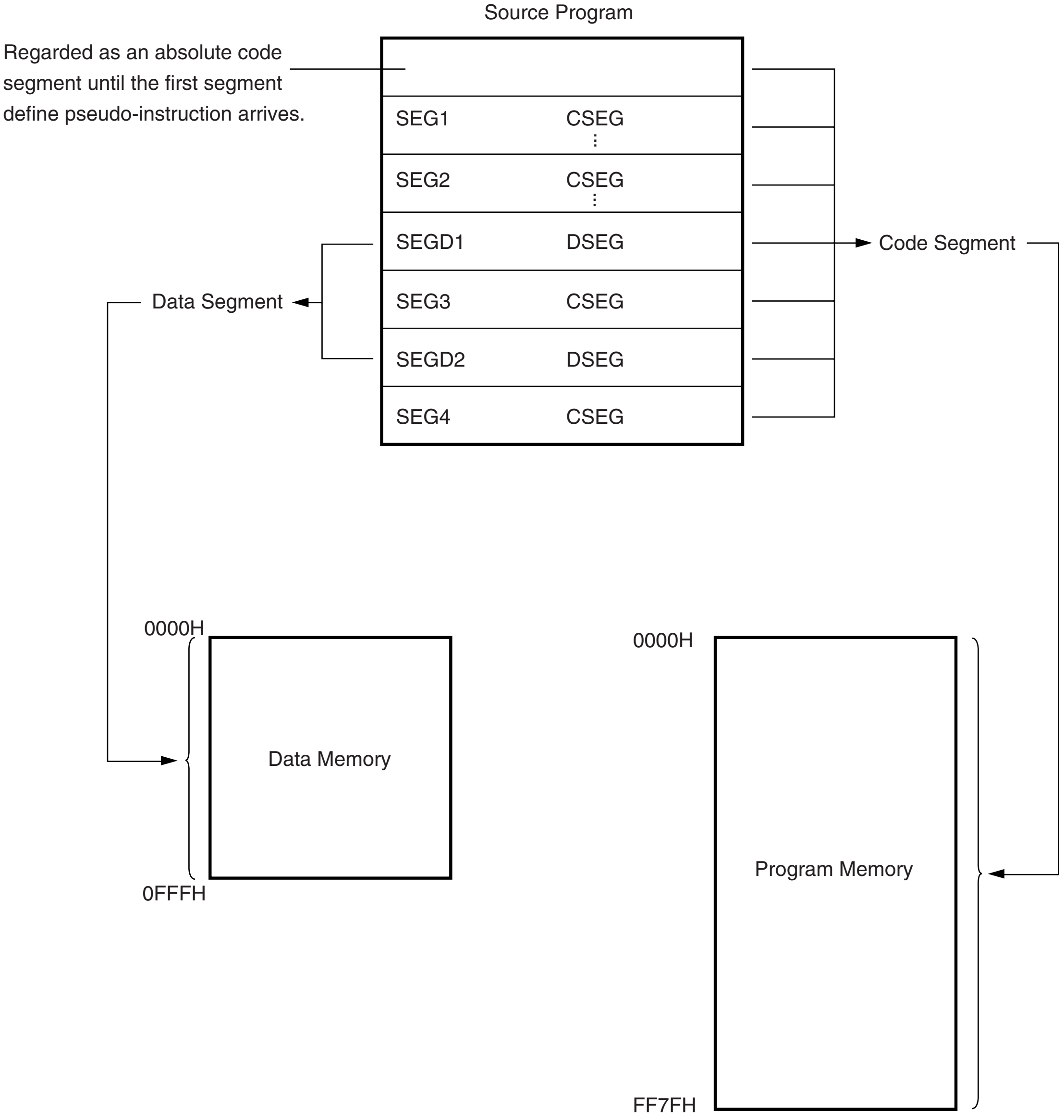
A group of source program statements defined for the use of program memory area by the CSEG pseudo-instruction is called ‘code segment’ and a group of source program statements defined for the use of data memory area by the DSEG pseudo-instruction is called ‘data segment’.

Figure 2-2 shows the Source Program Configuration.

The relocatable assembler regards an area up to where the first segment definition pseudo-instruction appears in the source program (even if the segment definition pseudo-instruction has not been described anywhere) as the code segment starting at address 10H (because program memory addresses 0 to 0FH might be used as the interrupt vector area).

As described above, the relocatable assembler source program consists of segments.

Figure 2-2. Source Program Configuration



2.3 PROGRAM MEMORY AND CODE SEGMENT

2.3.1 Reason why Code Segments are Relocatable

The 75X Series/75XL Series has a maximum program memory space of 64 Kbytes.

Greater than one programmer may be engaged in the development of such a large program. In view of development efficiency, maintenance and reliability, it may be desirable for greater than one person to be engaged in programming by dividing the program. This is called ‘modular programming’.

However, when executing the modular programming, the location in the program memory where each program is positioned cannot be clearly known until the programs of all persons involved are completed. Further, it is difficult to determine at the start of one program generation where the program should be positioned in the program memory space.

For these reasons, the code segments defined for program memory area use by the CSEG pseudo-instruction must be assembled so that they can be located anywhere in the program memory. This is why the code segments are relocatable.

2.3.2 Roles of Linker Relating to Relocation

The relocatable assembler assembles the relocatable code segments in relative address format setting the start address of one relocatable code segment at address 0H. By applying one of the assembly results or two or more object modules (object output in one-assembly units) to the linker, the relocatable code segments in each object module are relocated and the absolute addresses are determined.

As such the linker is a program to relocate the relative address object modules and replace them with the absolute address object modules.

However, the 75X series/75XL series program memoryhas special areas used for special purposes and structural boundaries defined by instruction restriction.

Thus, it is necessary for the programmer to understand the structural features of the program memory and to instruct the linker about the relocation positions of the relocatable code segments.

Remark The structural boundaries consist of a boundary called ‘block’ and a boundary called ‘page’.

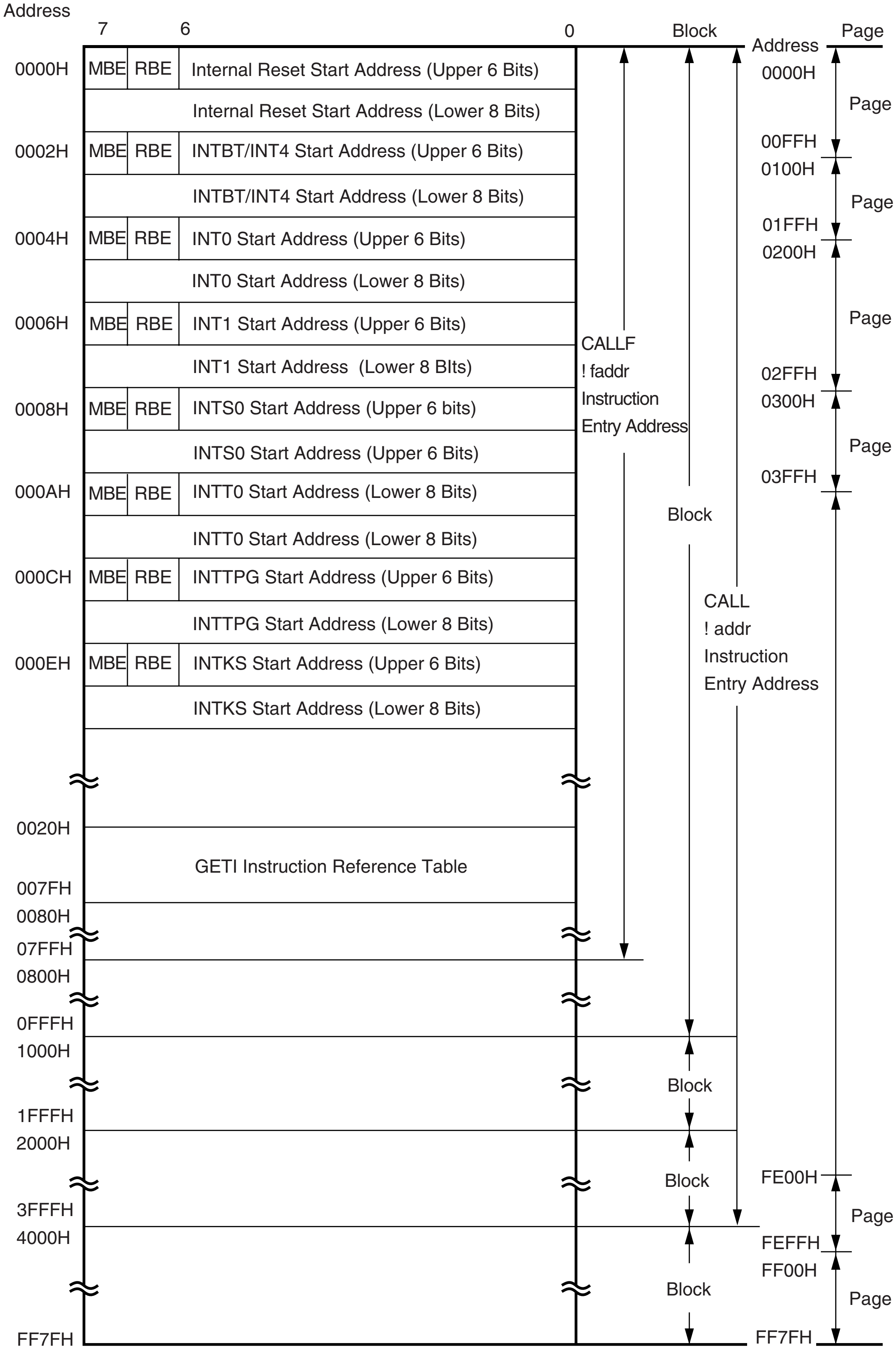
2.3.3 Structural Features of Program Memory

Figure 2-3 shows the special areas used for the specific purposes of the 75X series/75XL series program memory.

Addresses 0000H to 000FH : Vector table area for setting each vectored interrupt start address
Addresses 0020H to 007FH : GETI instruction reference table area
Addresses 0000H to 07FFH : Entry area for subroutines referred to by CALLF instruction
Addresses 0000H to 3FFFH : Area which can be branched by BR! instruction and entry area for subroutines referred to by CALL! instruction

Caution The vector table area varies depending on the product type.

Figure 2-3. Program Memory Map


Address
7
6
0
0000H
MBE RBE Internal Reset Start Address (Upper 6 Bits)
Internal Reset Start Address (Lower 8 Bits)
0002H
MBE RBE INTBT/INT4 Start Address (Upper 6 Bits)
INTBT/INT4 Start Address (Lower 8 Bits)
0004H
MBE RBE INT0 Start Address (Upper 6 Bits)
INT0 Start Address (Lower 8 Bits)
0006H
MBE RBE INT1 Start Address (Upper 6 Bits)
INT1 Start Address (Lower 8 BIts)
0008H
MBE RBE INTS0 Start Address (Upper 6 bits)
INTS0 Start Address (Upper 6 Bits)
000AH
MBE RBE INTT0 Start Address (Lower 8 Bits)
INTT0 Start Address (Lower 8 Bits)
000CH
MBE RBE INTTPG Start Address (Upper 6 Bits)
INTTPG Start Address (Lower 8 Bits)
000EH
MBE RBE INTKS Start Address (Upper 6 Bits)
INTKS Start Address (Lower 8 Bits)
0020H
GETI Instruction Reference Table
007FH
0080H
07FFH
0800H
0FFFH
1000H
1FFFH
2000H
3FFFH
4000H
FF7FH
CALLF
! faddr
Instruction
Entry Address
Block
Block
CALL
! addr
Instruction
Entry Address
Block
Block
Block
Address
0000H
00FFH
0100H
01FFH
0200H
02FFH
0300H
03FFH
FE00H
FEFFH
FF00H
FF7FH
Page
Page
Page
Page
Page
Page
Page

Caution The vector table area differs from the last address depending on the product type.

(1) Block

When the 64-Kbyte program memory is divided every 4K (4096) bytes from address 0H as shown in **Figure 2-3**, one 4-Kbyte unit is called ‘block’ and the boundary of two neighboring blocks is called ‘block boundary’.

The ‘block’ is the concept which derived from the character of the 75X Series/75XL Series branch instruction (BRCB instruction).

The BRCB instruction is a 2-byte branch instruction and is branched with the least significant 12 bits of the address described in the instruction operand replaced with those of the program counter. The most significant 2 bits of the program counter are the same as those of the current location address +2. Thus, the branch range varies depending on where the BRCB instruction is located in the program memory. This is the reason why the concept of ‘block’ has been introduced to the program memory of the 75X Series/75XL Series.

(2) Page

When the 64-Kbyte program memory is divided every 256 bytes from address 0H as shown in **Figure 2-3**, one unit (256 bytes) is called ‘page’ and the boundary of two neighboring pages is called ‘page boundary’.

The ‘page’ is the concept which derived from the characters of the 75X Series/75XL Series table reference instruction MOVT and branch instructions BR PCDE and BR PCXA.

The MOVT instruction is used to refer to the program memory table data. The DE or XA register contents are set to the least significant 8 bits of the program counter and the program memory contents addressed by the input DE or XA register contents are transferred to the XA register.

For example, when the program counter value is 100H and DE = 10H, data at program memory address 110H is transferred to the XA register.

The BR PCDE and BR PCXA branch instructions set the DE and XA register contents to the least significant 8 bits of the program counter. That means that the branch destination varies depending on the data transferred to the DE and XA register.

For example, when the program counter value is 5FFH and DE = 01H, the program counter value is changed to 501H.

In other words, the MOVT instruction reference destination and the BR PCDE and BR PCXA instruction branch destinations are both limited to inside the same page where those instructions are located. This is the reason why the concept of ‘page’ has been introduced to the program memory.

2.4 DATA MEMORY SPECIAL AREAS

The 75X Series/75XL Series data memory is provided with the peripheral hardware including input/output ports and timers at addresses 0F80H to 0FFFH. Thus, the absolute address specification of data segment is not possible for this area.

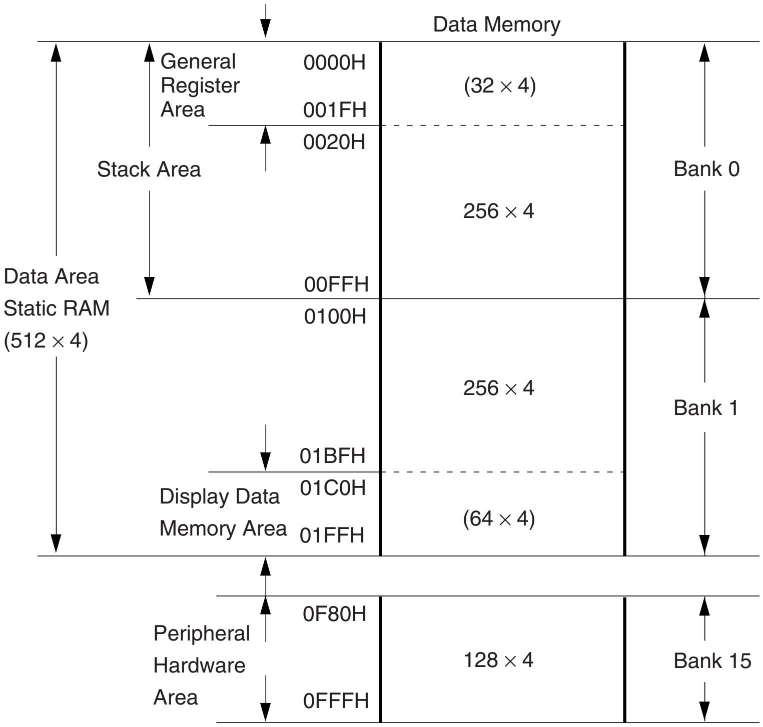
The assembler has the symbol (name) indicating the address of the hardware loaded at addresses 0F80H to 0FFFH as the reserved word. The reserved word is called ‘specific address name code’.

When operating the peripheral hardware, the specific address name code assigned for the address to be manipulated is described for the instruction operand.

Among the specific address name codes, the symbol attribute of the names indicating the bit addresses between 0FB0H.0 and 0FBFH.3 or 0FF0H.0 and 0FFFH.3 is ‘PBIT’ and the symbol attribute of all other names is ‘DATA’.

For details of symbol attributes, refer to **3.5 OPERAND CHARACTERISTICS**.

Figure 2-4. Data Memory Map



Caution **The area where hardware is incorporated depends on the product type.**
Depending on the product type, the stack area of memory banks other than memory bank 0 is also available.
★ **For how to set a value to the stack pointer, refer to 4.5 (3) STKLN.**

CHAPTER 3 SOURCE PROGRAM DESCRIPTION METHOD

In this chapter, the contents necessary for describing the source program (description format, formulas and operators, operand characteristics, etc.) will be described.

3.1 BASIC CONFIGURATION OF SOURCE PROGRAM

As described in **CHAPTER 2 75X SERIES/75XL SERIES FEATURES**, the 75X Series/75XL Series source program is configured in units called ‘segment’.

The segment is generally configured of functionally similar types of routine or data.

There are code segments and data segments, and they are located in the program memory (ROM) area and the data memory (RAM) area, respectively.

The segment memory space varies depending on the assembled product type.

For details, refer to **APPENDIX A LIST OF ASSEMBLED RELEVANT UNIT TYPES**.

The segment must be less than each memory space in size.

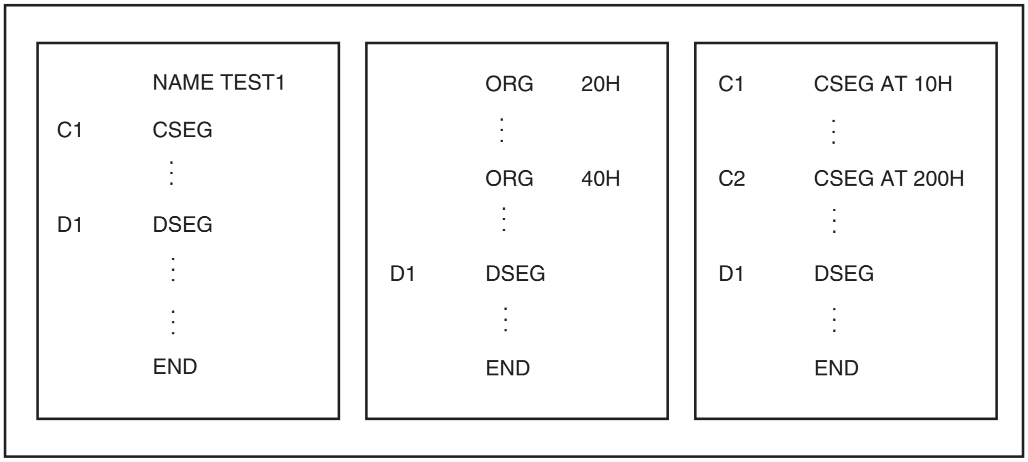
The code segment can be positioned at any address using the linker. The absolute code segment and the data segment cannot be changed from the address specified by the source program.

To provide instructions to the assembler in the source program, the appropriate object module is generated by placing assembler options or pseudo-instructions.

The source program can be configured by combining any segments.

Figure 3-1 shows a source program configuration example.

Figure 3-1. Source Program Configuration Example



```
        NAME TEST1
C1      CSEG
          :
D1      DSEG
          :
          :
        END
```

```
        ORG     20H
          :
        ORG     40H
          :
D1      DSEG
          :
        END
```

```
C1      CSEG AT 10H
          :
C2      CSEG AT 200H
          :
D1      DSEG
          :
        END
```

3.2 SAMPLE PROGRAM

This section shows a source program (source module) description example (this example is attached to the product as the sample program file).

Source module description procedure should be learned from this example.

This program is designed for the μPD75106.

The sample program contents are briefly explained below.

This sample program is an analog-to-digital conversion program to sample the analog signal (PTH00 pin input signal) eight times using the hardware (programmable threshold port and serial interface) incorporated into the μPD75106 and to generate the average value from the serial output pin.

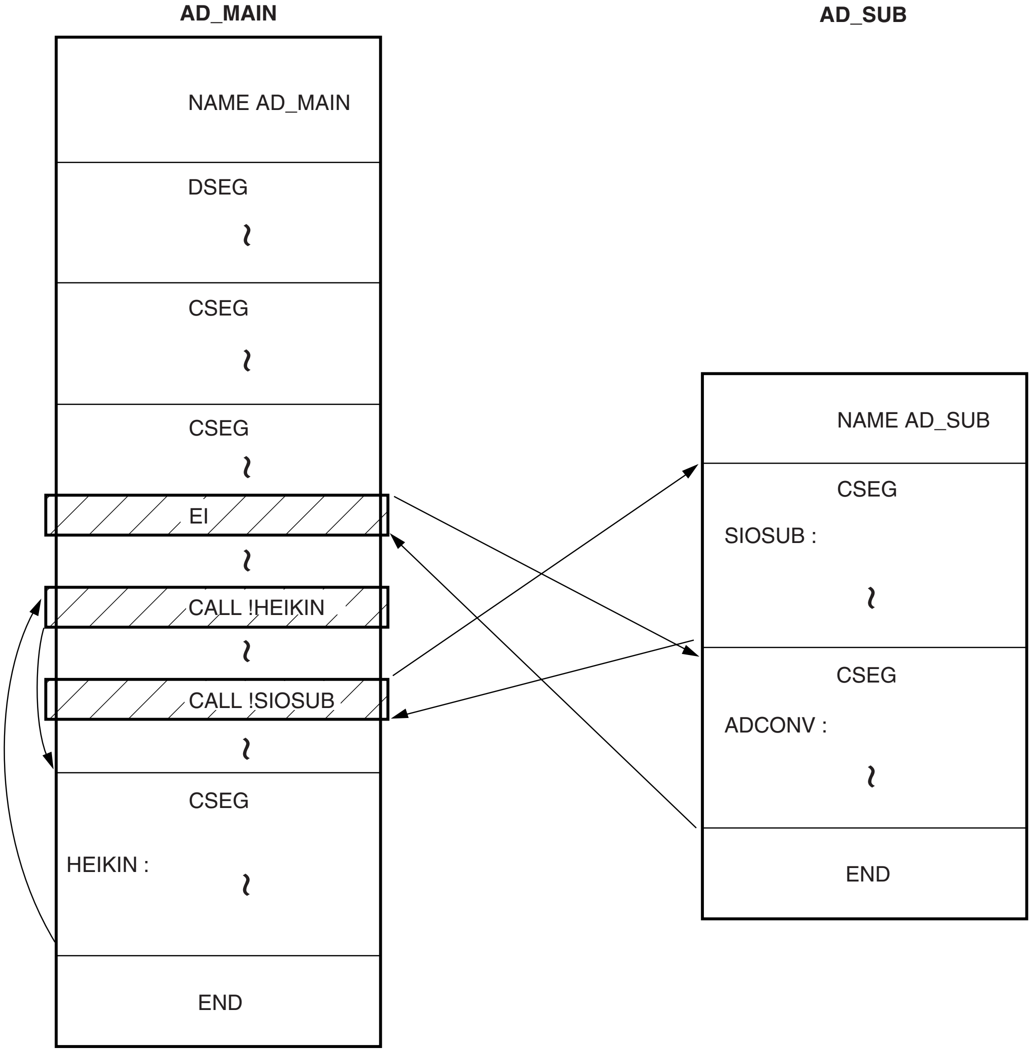
In this sample program, one program is divided into two modules.

One module is called AD_MAIN and stored in the source module file “75XTEST1.ASM”.

The other module is called AD_SUB and stored in the source module file “75XTEST2.ASM”.

Figure 3-2 shows the sample program configuration.

Figure 3-2. Sample Program Configuration



Caution

This sample program has been provided to learn the assembler package functions and operations. Thus, it cannot be used as an application program as it is.

<Main Routine>

```
$         TITLE = 'A-D CONVERT'                      ; (1)
; ********************************************
; ***     A-D CONVERT PROGRAM        ***
; ********************************************
          NAME      AD_MAIN                          ; (2)
          EXTRN     CODE (SIOSUB, ADCONV)            ; (3)
          PUBLIC    TDATA, SEL15                     ; (4)
          STKLN     10                               ; (5)
          VENT0     MBE = 1, RBE = 1, MAIN           ; (6)
          VENT4     MBE = 1, RBE = 0, ADCONV
SEG0      DSEG      1 AT 10H                         ; (7)
TDATA :   DS        2

; ***     GETI TABLE          ***
SEG1      CSEG      IENT                             ; (8)
SEL15 :   SEL       MB15

; ***     MAIN ROUTINE        ***
SEG2      CSEG      INBLOCK                          ; (9)
MAIN :    SEL       RB1

          GETI      SEL15              ; STACK POINTER SET
          MOV       XA, #STACK         ;
          MOV       SP, XA             ;

          MOV       A, #0011B
          MOV       PCC, A             ; PCC  ← 0011B

; **      DATA RAM 0H-13FH ZERO CLEAR     **

          SEL       MB1
          MOV       HL, #3FH
          MOV       XA, #00
LOOP1 :   MOV       @HL, A             ; 100H-13FH
          DECS      HL
          BR        LOOP1
          SEL       MB0
LOOP2 :   MOV       @HL, A             ; 0H-FFH
          DECS      HL
          BR        LOOP2
```

(1) Assembler option

(2) Module name declaration

(3) Declaration as an external reference symbol of the symbol defined by another module

(4) Declaration as an external definition symbol of the symbol to be referred to from another module

(5) Stack size specification

(6) Specification of the memory bank and register bank processing start addresses upon interruption

(7) Data segment start declaration

(8) Code segment start declaration

(9) Code segment start declaration

(10) Code segment start declaration

(11) Module end declaration

```
; **       TIMER SET (SAMPLING TIME = 30MSEC, FXX = 4. 19MHz)  **

           GETI      SEL15            ; SEL   MB15
           MOV       XA, #79H
           MOV       TMOD0, XA
           MOV       XA, #01001100B
           MOV       TM0, XA
           EI
           EI        IET0

           SEL       MB1
LOOP3 :    MOV       XA, #0H
           MOV       B, #00H
LOOP4 :    SKE       B, #08H
           BR        LOOP4
           CALL      !HEIKIN
           MOV       TDATA, XA
           CALL      !SIOSUB
           BR        LOOP3

; ***      HEIKIN    (SAMPLE NUMBERS = 8)  ***
SEG3       CSEG      SENT                                 ; (10)
HEIKIN :   MOV       C, #2H
LOOP5 :    XCH       A, X
           CLR1      CY
           RORC      A
           XCH       A, X
           RORC      A
           DECS      C
           BR        LOOP5
           RET

           END                                            ; (11)
```

<Subroutine>

```
$         TITLE = ‘A-D CONVERT’                              ; (12)
; ******************************************
; ***     A-D CONVERT PROGRAM          ***
; ******************************************
          NAME      AD_SUB                                   ; (13)
          EXTRN     DATA (TDATA), CODE (SEL15)               ; (14)
          PUBLIC    SIOSUB, ADCONV                           ; (15)
          STKLN     2                                        ; (16)
; ***     SIO SUB-ROUTINE    ***
SEG4      CSEG      SENT                                     ; (17)
SIOSUB :  PUSH      BS
          SEL       RB2
          SEL       MB1
          MOV       XA, TDATA
          GETI      SEL15           ; SEL     MB15
          MOV       SIO, XA
          MOV       XA, #11101110B
          MOV       SIOM, XA        ; CLOCK = 262KHZ, MSB
          POP       BS
          RET

; ***     ANALOG INPUT    (RBE = 0)    ***
SEG5      CSEG      SENT                                     ; (18)
ADCONV :  PUSH      BS
          GETI      SEL15           ; SEL     MB15
          MOV       HL, #0D3H
          MOV       XA, #0C0H
          MOV       BSB0, A         ; BSB0 ← 0H
LOOP :    SET1      BSB0, @L
          MOV       A, BSB0
          MOV       PTHM, XA        ; COMP. START
          MOV       A, #0AH         ; 18 MACHINE
WAIT :    INCS      A               ; CIRCLE WAIT
          BR        WAIT
          MOV1      CY, @H+PTH0, 0
          MOV1      BSB0. @L, CY
          DECS      L
          BR        LOOP
          MOV       X, #0H
          MOV       A, BSB0
          ADDS      XA', XA         ; ADD DATA
          SET1      RBE
          POP       BS
          INCS      B               ; SAMPLE COUNT INC.
          RETI

          END                                                ; (19)
```

(12) Assembler option

(13) Module name declaration

(14) Declaration as an external reference symbol of the symbol defined by another module

(15) Declaration as an external definition symbol of the symbol referred to from another module

(16) Stack size specification

(17) Code segment start declaration

(18) Code segment start declaration

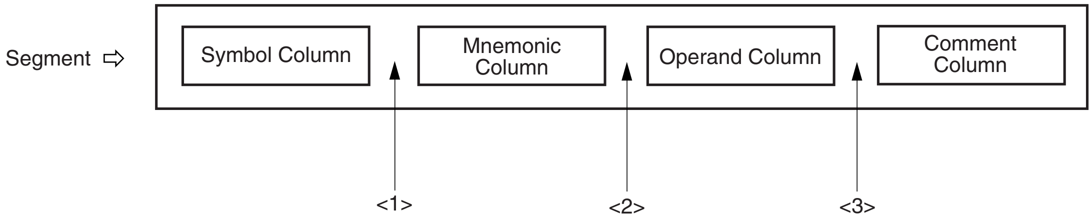
(19) Module end declaration

3.3 SOURCE PROGRAM DESCRIPTION FORMAT

3.3.1 Statement Format

The source program consists of statements.
One statement consists of 4 fields indicated in **Figure 3-3. Statement Component Fields**.

Figure 3-3. Statement Component Fields



<1> The symbol column and the mnemonic column are divided with a colon (:) or more than one blank space (or TAB).
★ The separator to be used, a colon or space character, depends on the instruction described in the mnemonic column.

<2> The mnemonic column and the operand column are divided with more than one blank space (or TAB). The operand column may not be necessary depending on the instruction to be described in the mnemonic column.

<3> When entering in the comment column, describe a semi-colon (;) before the comment column.

One statement is described on one line.
Using description procedure based on a free method, description can be started with any column in the order of the symbol, mnemonic, operand and comment columns.

Description can be done for the following lines:

- Blank line (line having no statement description)
- Line with symbol column only
- Line with comment column only

3.3.2 Character Set

For statement description, standard-size alphanumeric characters and standard-size special characters are used (editors etc. available on the market may be used).

(1) Alphabetic letters (Figures in parentheses represents the JIS code.)

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
(41H)	(42H)	(43H)	(44H)	(45H)	(46H)	(47H)	(48H)	(49H)	(4AH)	(4BH)	(4CH)	(4DH)	(4EH)	(4FH)	(50H)	(51H)
R	S	T	U	V	W	X	Y	Z								
(52H)	(53H)	(54H)	(55H)	(56H)	(57H)	(58H)	(59H)	(5AH)								
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
(61H)	(62H)	(63H)	(64H)	(65H)	(66H)	(67H)	(68H)	(69H)	(6AH)	(6BH)	(6CH)	(6DH)	(6EH)	(6FH)	(70H)	(71H)
r	s	t	u	v	w	x	y	z								
(72H)	(73H)	(74H)	(75H)	(76H)	(77H)	(78H)	(79H)	(7AH)								

Caution

When a reserved word is described using small alphabetic letters, they are interpreted as capital letters.

(2) Numerals (Figures in parentheses represents the JIS code.)

0	1	2	3	4	5	6	7	8	9
(30H)	(31H)	(32H)	(33H)	(34H)	(35H)	(36H)	(37H)	(38H)	(39H)

(3) Special characters

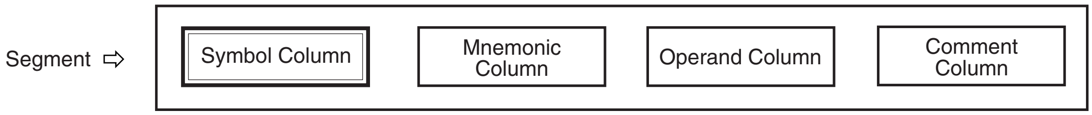
Characters	JIS Code	Name	Main Applications	
?	3FH	Question mark	Character corresponding to alphabetic letter	
@	40H	Unit price symbol	Indirect addressing start symbol	
_	6FH	Underline	Character corresponding to alphabetic letter	
	20H	Blank	Division symbol	Division symbol foreach column
HT	09H	Tab code		Character corresponding to blank space
,	2CH	Comma		Operand division symbol
:	3AH	Colon		Label division symbol
;	3BH	Semi-colon		Comment column start symbol
CR	0DH	Return code		Last symbol of one line (ignored by the assembler)
LF	0AH	Line feed code		
+	2BH	Plus	Assembler operator	Add operator or plus sign
–	2DH	Minus		Subtraction operator or minus sign
*	2AH	Asterisk		Multiplication operator
/	2FH	Slash		Division operator
.	2EH	Period		Bit operator
( )	28H	Left and right parentheses		Operation order change
’	27H	Quotation mark	Character constant start and end symbols	
<	3CH	Inequality sign	Compare operators	
>	3EH			
=	3DH	Equality sign		
$	24H	Dollar symbol	• Location counter value • Assembler option start symbol • Relative addressing specification symbol	
#	23H	Sharp	• Immediate addressing specification symbol	
!	21H	Exclamation	• Absolute addressing specification symbol • Illegal character representation in the assembly list	
NULL	00H	Null code	Ignored by the assembler.	
FF	0CH	Form feed		
DEL	FFH	Delete code		
&	26H	Ampersand	Linkage between macro parameter and character string	

Remark Characters of codes 80H to FFH can be described in the comment column only. Comments can be entered using kanji.

3.3.3 Character Configuration Fields

This section describes the character component fields.

(1) Symbol column



A symbol is described in the symbol column. The symbol is a name for the numeric value data or address.
Using the symbol makes it easy to understand the source program contents.
The symbol types, attributes and description rules are shown below.

[Symbol types]

Table 3-1 gives a list of symbol types according to purposes and definitions.

Table 3-1. Symbol Types

Symbol Type	Purpose	Definition
Name	Used as numeric value data in the source program.	Described in the symbol column of EQU, SET, CSEG and DSEG pseudo-instructions or the operand column of the EXTRN pseudo-instruction.
Label	Used as address data in the source program.	Described in the symbol column of instructions and ORG, DB, DS, BR, VENTn, TCALL and TBR pseudo-instructions. Colon (:) is used as the division symbol.
Segment Name	Used as the operation target in linker option.	Described in the symbol column of the CSEG and DSEG pseudo-instructions.
★ Macro Name	Used as the expansion location of the macro.	Described in the symbol column of the MACRO instruction.

[Symbol description rules]

The symbols are described according to the following rules:

<1> Each symbol consists of alphanumerics and characters corresponding to alphabetic letters (?, _). Numerals (0 to 9) cannot be used as the start character.

★ <2> Each symbol can have a length of 1 to 31 characters (when -NS option is specified: 1 to 8 characters) (for details of -NS option, refer to **Operation** Manual). It is all right if symbols with 32 (9) or more characters are described, however, only the first thirty one characters (eight characters) are valid.

<3> The reserved word cannot be used as a symbol. The reserved words are shown in **APPENDIX B LIST OF RESERVED WORDS**.

<4> The same symbol cannot be defined more than twice. (The name defined by the SET pseudo-instruction can be redefined by the SET pseudo-instruction.)

<5> If a symbol is described with small alphabetic letters, they are interpreted as capital alphabetic letters.

<6> The label and the mnemonic column are divided by colon (:) and the name and the mnemonic column are divided by a blank space.

<7> Only one label can be described on one line.

Example 1. Correct symbols

```
TEN         EQU     10H      ;  'TEN' is a name.
NEXT:       BR      !100H    ;  'NEXT' is a label.
C1          CSEG             ;  'C1' is a segment name.
```

2. Incorrect symbols

```
1ST:        MOV     A, #0H   ;  Numeral cannot be used for the start character.
TEN:        EQU     10H      ;  'TEN' is a name.  Colon (:) is not necessary.
NEXT        BR      !100H    ;  'NEXT' is a label.  The label and the mnemonic column are divided
                                using colon (:).
TEN         EQU     10H      ;  'TEN' and 'ten' are the same name symbols.
ten         EQU     10H         Thus, the description 'ten' is an error.
```

3. Statement consisting of symbols only

```
ABCD:                        ;  'ABCD' is defined as a label.
```

4. Others

```
ABC         EQU        3     ;  The same data "3" is assigned for ABC and XYZ.
XYZ         EQU        ABC
LOOP :      ADDC       A, @HL
              :
              :
LOOP :      MOV        A, B
              :
              :
            BR         $LOOP
An error results because label "LOOP" has been doubly defined.
Example of more than one description on the same line
MAIN : FLY : LOOP : MOV A, B ...  An error results.
```

[Symbol attributes]

The name and label have values and attributes.

The module and macro names have no values.

The value is a defined numeric value data or address data value.

The attributes are symbol attributes listed below.

Table 3-2. Symbol Attribute Types

Attribute Type	Description
NUMBER	○ Name with the constant (except specific address name codes and bit values) defined using EQU and SET pseudo-instructions ○ External reference name with the symbol attribute declared as ‘NUMBER’ using EXTRN pseudo-instruction
CODE	○ Label defined in the code segment ○ Name with the label having symbol attribute ‘CODE’ or ‘$’ (location) in the code segment defined using EQU and SET pseudo-instructions ○ External reference name with the symbol attribute declared as ‘CODE’ using EXTRN pseudo-instruction
DATA	○ Label defined in the data segment ○ Name with the label having symbol attribute ‘DATA’ or ‘$’ (location) in the data segment defined using EQU and SET pseudo-instructions ○ External reference name with the symbol attribute declared as ‘DATA’ using EXTRN pseudo-instruction ○ Specific address name code (except bit values) ○ Reserved word STACK (the symbol attributes are included in DATA attributes although displayed as STACK)
BIT	○ Name with the bit value defined using EQU and SET pseudo-instructions (The name with the bit value defined using a specific address name code and the symbol value set to FB0H.0 to FBFH.3 or FF0H.0 to FFFH.3 becomes the following PBIT attribute.) ○ Specific address name code with the bit value except FB0H.0 to FBFH.3 or FF0H.0 to FFFH.3 ○ External reference name with the symbol attribute declared as ‘BIT’ using EXTRN pseudo-instruction
PBIT	○ Specific address name code with the bit value of FB0H.0 to FBFH.3 or FF0H.0 to FFFH.3 ○ Symbol with the bit value defined using a specific address name code and the symbol value of FB0H.0 to FBFH.3 or FF0H.0 to FFFH.3 ○ External reference name with the symbol attribute declared as ‘PBIT’ using EXTRN pseudo-instruction

Example

```
  TEN      EQU   10       ;  Name ‘TEN’ has NUMBER attribute and value 10.
  SEG0     CSEG           ;  ‘SEG0’ is a segment name.
  DATA :   DB    0AH      ;  Label ‘DATA’ has CODE attribute and value 0AH.
  D1       DSEG           ;  ‘D1’ is a segment name.
  WORK :   DS    5        ;  Label ‘WORK’ has DATA attribute and value 5.
  BIT1     EQU   0FE0H.2  ;  Name ‘BIT1’ has BIT attribute and value 0FE0H.2.
★ PBIT1    EQU   PORT0.0  ;  Name ‘PBIT1’ has PBIT attribute and same value as PORT0.0.
```

The symbol attribute names are shown below.

Table 3-3. Symbol Attribute Names

Item No.	Symbol Attribute	Name
1	NUMBER	Constant symbol
2	CODE	Code symbol
3	DATA	Data symbol
4	BIT	Bit symbol
5	PBIT	Port bit symbol

[Necessity of symbol attributes]

In the following example, the symbol attribute of labels D1 and D2 is ‘DATA’ and the symbol attribute of label C1 is ‘CODE’.

Take a look at (a). BRCB instruction is a 2-byte branch instruction indicating that the operand value is set to the least significant 12 bits of the program counter and is branched to the program memory. In the case of (a), however, the operand value is the data memory address. This is a program error.

Example

```
; **     DEFINE WORKING AREA    **
DATA1       DSEG    1 AT 0H
D1 :        DS      2
D2 :        DS      2
; **    MAIN PROGRAM    **
CODE1       CSEG
C1 :        MOV     A, B
              ≀
            BR      C1
            MOV     XA,D2
              ≀
            BRCB    D1          ; (a)
              ≀
            END
```

The symbol described as each instruction operand must have an attribute so that the assembler can detect such program errors.

In the case of (a), the assembler checks if the symbol attribute (NUMBER or CODE) required by BRCB instruction as an operand matches the symbol attribute of symbol ‘D1’ actually described in the operand column. Because the result shows that the symbol attribute required by BRCB instruction as an operand is not ‘DATA’, the assembler detects an error and generates the relevant error message in the list.

As explained above, program description errors can be prevented from occurring by the assembler providing all symbols described in the source program with symbol attributes. This is the reason why attributes are necessary for the symbols.

Caution

> The name indicating the bit address with a specific address name code has special attribute called ‘PBIT’. Thus, when operating the peripheral hardware, operands can be checked more precisely by describing specific address name codes.

Since the symbol attributes are necessary to understand the EXTRN pseudo-instruction described later, you should learn by heart the symbol attribute types and which symbols have symbol attributes.

★ **[Valid range of symbol in macro]**

The symbol defined in a macro is usually valid only within the macro (called local symbol). Therefore, a local symbol has no relationship with symbols located outside the macro or in other macros.

On the other hand, a symbol referenced by plural macros or used in common between a macro and a statement outside the macro is called global symbol. Global symbols are useful when common values must be set or referenced from one another.

The next section explains the valid ranges of global symbol and local symbol that are used in macros.

<1> Global symbol

A symbol declared with the GLOBAL instruction for its label and SET name will be treated as a global symbol.

Upon GLOBAL declaration, a global symbol becomes valid in the entire source program. If the declaration is made within a macro definition part, the symbol will become valid when and after the macro is developed.

<2> Local symbol

Unless otherwise declared, the symbol in a macro is automatically defined as a local symbol. Therefore, it cannot be referenced from outside the macro or used in common by plural macros.

If a macro body includes a label (the description format is “SymbolName:”), the symbol name is changed and output to the output list file because the assembler’s symbol description rules prohibit labels with an identical name. In the output process, such name will be changed as follows:

(a) The modified symbol name is Znnnnn, where Z is the first character of the original symbol name and nnnnn is an appended incrementing number between 00000 and 65535 (in decimal notation).
If nnnnn reaches 65535, the next number returns to 00000.

(b) Each time a macro references the symbol, nnnnn is incremented by 1 to create a new symbol name.

(c) If a macro includes a label, the name of the symbol which references the label name in the macro will also be modified to Znnnnn at the same time.
However, the nnnnn value does not change for the same label name.

(d) If, within a macro, a label is named the same as a symbol name which has been already defined in the SET statement as a local symbol, an error message is output. Then, such label will be output to the output list file in a source program image when the macro is developed.
The local symbol’s SET is not output when the macro is developed. If the local symbol defined in a SET definition statement is referenced by a statement in the macro body, the result is shown below.

(i) When it is described in the macro instruction operand formula:
The value the symbol includes is referenced with its local symbol name unchanged.

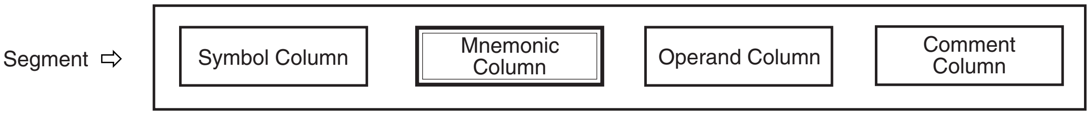
(ii) In other cases:
If the symbol is numeric, only the value the symbol includes is converted directly to a decimal number and then output to the output list file.

If the macro is nested, a symbol defined in the macro or in a lower nesting level macro involving the macro can be referenced.

An identical name cannot be used for the SET name and the label in the same macro.

If an identical name is used, an error occurs when the macro is defined and the symbol name will be regarded as a mere character string when the macro is referenced.

(2) Mnemonic column



The instruction mnemonic, pseudo-instruction and control instruction are described in the mnemonic column. In the case of an instruction or a pseudo-instruction requiring an operand, the mnemonic and operand columns are divided using more than one blank space or TAB.

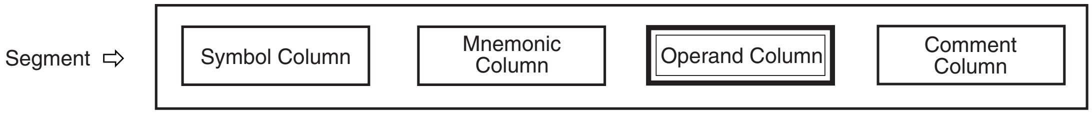
Example 1. Correct example

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

2. Incorrect example

```
MOVA,       #0H                ;  There is no blank space between the mnemonic column and the operand
                                  column.
CAL L       !CONVAH            ;  There is a blank space in the mnemonic.
HLT                            ;  There is no 'HLT' in the 75X Series/75XL Series instructions.
```

(3) Operand column



Data necessary for execution of an instruction or a pseudo-instruction is described in the operand column.

Some instructions and pseudo-instructions may not require operands or may require more than one operand. When describing more than one operand, each operand is divided with a comma (,).

The following items can be described in the operand column.

- Constant (numeric constant and character constant)
- Register name
- Special character ($ # ! @)
- Name and label
- Formula
- Specific address name code

The required operand size and symbol attributes vary depending on the instruction and pseudo-instruction. For details, refer to **3.5 OPERAND CHARACTERISTICS**.

Each item describable in the operand column is explained below.

[Constants]

Each constant has its own value. There are numeric and character constants.

- **Numeric constant (immediate data)**

  Binary, octal, decimal and hexadecimal numbers can be described as numeric constants. Procedure for representing each numeric constant is as follows:

Table 3-4. Numeric Constant Representation

Numeric Constant Type	Representation	Example
Binary number	• Add ‘B’ at the end of numeric value.	1101B
Octal number	• Add ‘O’ at the end of numeric value.	74O
Decimal number	• Describe the numeric value what it is. • Or add ‘D’ at the end.	128 128D
Hexadecimal number	• Add ‘H’ at the end of numeric value. • If the first letter is ‘A’, ‘B’, ‘C’, ‘D’, ‘E’ or ‘F’, add ‘0’ before it.	8CH 0A6H

- **Character constant**

  The character constant is the characters in **3.3.2 Character Set** marked using quotation marks (’). It is assembled and converted to a 7-bit ASCII code with a parity bit set to 0.

  When using the quotation marks, describe two quotation marks continuously.

  Character constant representation example

  ‘A’
  ‘ ’ ; one blank space
  ‘ ’ ‘ ’ ; one quotation mark
  ‘main’

[Register name]

Registers listed in **Table 3-5. Register Types** can be described in the operand column.

Table 3-5. Register Types

Register Name	Description Format
General register	A, B, C, D, E, H, L, X
Register pair	XA, BC, DE, HL, XA', BC', DE', HL'
Special format	@BCDE, @BCXA, @PCDE, @PCXA, @DE, @DL, @HL, @HL+, @HL–, @H+mem.bit, pmem.@L
Special register	MBn, RBn, BS
Flag	CY

Cautions

1. Some registers listed in **Table 3-5. Register Types** may not be used depending on the assembled products. For details, refer to **APPENDIX A LIST OF ASSEMBLED RELEVANT UNIT TYPES** and **APPENDIX B LIST OF RESERVED WORDS**.
2. These register names cannot be used for formula (refer to **“Formula”**).

   Example MOV A, B+1 ... B+1 cannot be described.
3. As a special format, a blank space or TAB can be inserted between @ and BCDE as @ BCDE.

[Special character]

Special characters listed below can be described in the operand column.

Table 3-6. Special Characters Describable in Operand Column

Special Character	Function
$	• Indicates the location address (1st byte in the case of an instruction with greater than one byte) for which an instruction having this operand is assigned. • Indicates the relative addressing of the branch instruction.
!	• Indicates the absolute address of the branch and call instructions.
#	• Indicates immediate data.
@	• Indicates indirect addressing.

Example Usage example of special character

```
Address    Source program
100        LOOP :  INCS    A
101                BR      $-1   ... <1>
```

In the case of <1>, ‘$’ in the operand column indicates the location address 101 for which the 1st byte of the object code for instruction ‘BR $-1’. The description of <1> is rewritten as ‘BR LOOP’.

```
Source program
      BR     !100H   ;   '!' indicates the absolute addressing of the unconditional branch instruction.
      AND    A, #6H  ;   '#' indicates immediate data.

SIX   EQU    6H
      AND    A, #SIX ;   '#' indicates immediate data.

      AND    A, @HL  ;   '@' indicates indirect addressing.
```

[Name and label]

When a name or a label is described in the operand column, the name or label value is manipulated as numeric data by the instruction or pseudo-instruction.

Example 1. Usage example of name

```
SIX  EQU    6H
     MOV    A, #SIX ; This description can be rewritten as 'MOV A, #6H'.
```

2. Usage example of label

```
        ORG     100H
LOOP :  INCS    A
        BR      LOOP ; This description can be rewritten as 'BR 100H'.
```

[Formula]

A formula can be described in the operand column.

The formula combines constants, special characters and names or labels using operators. It can be described in a location where numeric representation is possible as an instruction operand.

Formulas and operators will be described in **3.4**.

(Formula description example)

```
SIX  EQU    6H
     MOV    A, #SIX-5H
```

In this example, #SIX-5H is the formula.

In this formula, the name and the numeric constant are combined using a minus operator (–). The formula value is 1H. Thus, this description is rewritten as 'MOV A, #1H'.

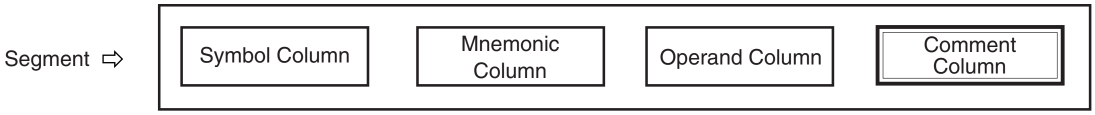
[Specific address name code]

The assembler has the symbols reserved to control the hardware such as input/output ports and timers at 0F80H to 0FFFH in the data memory space. These symbols are called specific address name codes.

The specific address name code differs depending on the assembled product type. For details, refer to **APPENDIX B LIST OF RESERVED WORDS**.

For the address and meaning of each specific address name code, refer to the User's Manual for each device.

(4) Comment column



In the comment column, comment is described after the semi-colon (;). An easy-to-understand source program can be generated by describing comment. Description in the comment column does not undergo assembling called machine word conversion and is output in the assembly list.

Characters listed in **3.3.2 Character Set** can be described in the comment column.

Example

```
$       TITLE = 'A-D CONVERT '
; ***********************************************          ┐
; ***   A-D CONVERT PROGRAM                     ***        │ Line with comment only
; ***********************************************          ┘
          NAME      AD_MAIN
          EXTRN     CODE (SIOSUB, ADCONV)
          PUBLIC    TDATA, SEL15
          STKLN     10
          VENT0     MBE = 1, RBE = 1, MAIN
          VENT4     MBE = 1, RBE = 0, ADCONV
SEG0      DSEG      1 AT 10H
TDATA :   DS        2
                                                           ┐ Line with comment only
; ***     GETI TABLE           ***                         ┘
SEG1      CSEG      IENT
SEL15 :   SEL       MB15

; ***     MAIN ROUTINE         ***
SEG2      CSEG      INBLOCK
MAIN :    SEL       RB1

          GETI      SEL15         ; STACK POINTER SET      ┐
          MOV       XA, #STACK    ;                        │
          MOV       SP, XA        ;                        │ Line with comment described
                                                           │ in the comment column
          MOV       A, #0011B                              │
          MOV       PCC, A        ; PCC ← 0011B            ┘
```

3.4 FORMULAS AND OPERATORS

The formula combines constants, special characters, names and labels with operators.

Formula component elements except operators are called terms. The terms in turn are called the first, second terms and so on from the left.

There are two types of formulas; absolute formula in which values are determined when assembled, relocatable formula in which values are determined when linked and external reference formula.

Table 3-7 gives a list of operator types and **Table 3-8** shows priority order concerning operation execution.

To change the operation order, parentheses '( )' are used.

Example

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

In <1>, 5* (SYM+1) is the formula. 5, SYM and 1 are the first, second and third terms, respectively. *, + and ( ) are operators.

Table 3-7. Operator Types

Operator Types	Operators
Arithmetic operators	+, –, *, /, MOD, + sign, – sign
Logical operators	NOT, AND, OR, XOR
Compare operators	EQ or =, NE or < >, GT or >, GE or >=, LT or < and LE or <=
Shift operators	SHR, SHL
Bit location specification operator	. (period)
Byte separation operators	HIGH, LOW
Others	( )

Table 3-8. Operator Priority Order

Operator Priority Order		Operator
High	1	. (bit location specification operator)
	2	HIGH, LOW
	3	+ sign, – sign, NOT
	4	*, /, MOD, SHR, SHL
	5	+, –
	6	AND
★	7	OR, XOR
★ Low	8	EQ, NE, GT, GE, LT, LE, =, < >, >, >=, <, <=

Formula operations are carried out according to the following rules:

<1> Operation order follows the operator priority order. If the operators have the same priority, the operation on the left is first carried out.

<2> Operation in parentheses '( )' is carried out ahead of operation outside parentheses.

<3> The external reference symbol undergoes operation with its value set to 0 when assembled. When linked, the correct value is assigned. For details, refer to **3.4.2 Operation Restrictions**.

<4> In division, decimal fractions are omitted.

<5> When the devisor is 0, the error is printed and the result is made 0.

<6> Negative numbers are in two's complement format.

3.4.1 Operator Functions

This section describes the operator functions.

(1) Arithmetic Operators

(a) + (add)

[Function]

Returns the sum of the 1st and 2nd term values.

[Usage Example]

```
C1          CSEG    AT 100H
START:      BR      $$ +6H                  ; (a)
             .
             .
             .
```

Branches to “address assigned for ‘START’ + address 6” by BR instruction. Namely, jumps to “100H + 6H = 106H”.

Thus, (a) can also be described as “START: BR $106H”.

(b) – (subtraction)

[Function]

Returns the balance between the 1st and 2nd term values.

[Usage Example]

```
C2          CSEG    AT 100H
BACK:       BR      $BACK-6H                ; (b)
             .
             .
             .
```

Branches to “address assigned for ‘BACK’ – address 6” by BR instruction. Namely, jumps to “100H – 6H = 0FAH”.

Thus, (b) can also be described as “START: BR $0FAH”.

(c) * (multiplication)

[Function]

Returns the product of the 1st and 2nd term values.

[Usage Example]

```
TWO        EQU    2H
           MOV    A, #TWO*3              ; (c)
              :
              :
```

Value 2H is defined for name ‘TWO’ by EQU pseudo-instruction.

Formula “TWO*3” means “2H*3” and value 6H is loaded into the A register.

Thus, (c) can also be described as “MOV A, #6H”.

(d) / (division)

[Function]

Divides the 1st term value by the 2nd term value and returns the integer part of the quotient.

The decimal fractions are omitted.

If the divisor is 0, an error results.

[Usage Example]

```
Y5:        MOV    A, #256/50             ; (d)
```

“256/50 = 5 with remainder 6” results.

Accordingly 5 in the integer part is loaded into the A register.

Thus, (d) can also be described as “MOV A, #5”.

(e) + sign

[Function]

Returns the term value what it is.

[Usage Example]

```
FIVE        EQU     +5
```

Returns the term value 5 what it is.

Value 5 is defined for name ‘FIVE’ by EQU pseudo-instruction.

(f) – sign

[Function]

Returns two’s complement of the term value.

[Usage Example]

```
NO          EQU     –1
```

Two’s complement of 0001B becomes 1111B.

Thus, value 0FH is defined for name ‘NO’ by EQU pseudo-instruction.

(g) MOD (remainder)

[Function]

Returns the remainder of the 1st term value divided by the 2nd term value.

A blank space is necessary before and after ‘MOD’.

When the divisor is 0, an error results.

[Usage Example]

```
REM6 :      MOV     A, #     256   MOD     50     ; (e)
```

“256/50 = 5 with remainder 6” results.

Accordingly, remainder 6 is loaded into the A register.

Thus, (e) can also be described as “MOV A, #6”.

(2) Logical Operators

(a) NOT (negation)

[Function]

Returns the bit-wise logical NOT of the term value.

A blank space is necessary between ‘NOT’ and the term.

[Usage Example]

```
COMPL :     MOV     XA, #NOT    0FFF3H          ;       (a)
```

Obtains the bit-wise logical NOT of 0FFF3H.

```
NOT)    1111    1111    1111    0011
-------------------------------------
        0000    0000    0000    1100
```

Accordingly, the value 0CH is loaded into the XA register.

Thus, (a) can also be described as “MOV XA, #0CH”.

(b) AND (logical product)

[Function]

Returns the bit-wise logical AND of the 1st and 2nd term values.

A blank space is necessary before and after ‘AND’.

[Usage Example]

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

Obtains the logical AND of 6FA and 0FH.

```
          0110    1111    1010
AND)      0000    0000    1111
-------------------------------
          0000    0000    1010
```

Accordingly, the value 0AH is loaded into the A register.

Thus, (b) can also be described as “MOV A, #0AH”.

Logical Operators **OR, XOR** **Logical Operators**

(c) OR (logical sum)

[Function]

Returns the bit-wise OR of the 1st and 2nd term values.

A blank space is necessary before and after ‘OR’.

[Usage Example]

```
MDFY1 :    MOV    A, #0AH    OR     1101B   ;     (c)
```

Obtains the OR of 0AH and 1101B.

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

Accordingly, the value 0FH is loaded into the A register.

Thus, (c) can also be described as “MOV A, #0FH”.

(d) XOR (exclusive logical sum)

[Function]

Returns the bit-wise, exclusive logical sum of the 1st and 2nd term values.

A blank space is necessary before and after ‘XOR’.

[Usage Example]

```
MDFY2 :    MOV    A, #0AH    XOR    1101B   ;     (d)
```

Obtains the exclusive OR of 0AH and 1101B.

```
       1010
XOR)   1101
       ----
       0111
```

Accordingly, the value 7H is loaded into the A register.

Thus, (d) can also be described as “MOV A, #7H”.

(3) Compare Operators

(a) EQ or = (equal)

[Function]

Returns 0FFFFH (true) when the 1st and 2nd term values are equal or 0000H (false) when those values are not equal.

A blank space is necessary before and after ‘EQ’.

[Usage Example]

```
A1      EQU     8H
A2      EQU     4H

        MOV     A, # (A1 EQ (A2+4))     AND 0FH ; (a)
        MOV     A, # (A1 EQ A2)         AND 0FH ; (b)
```

In the case of (a),

“A1 EQ (A2 + 4)” becomes “8H EQ (4H + 4)”.

Since the 1st and 2nd term values are equal, 0FFFFH is returned, the logical product with 0FH is obtained and 0FH is loaded into the A register.

In the case of (b),

“A1 EQ A2” becomes “8H EQ 4H”.

Since the 1st and 2nd term values are not equal, 0000H is returned, the logical product with 0FH is obtained and 0H is loaded into the A register.

(b) NE or < > (not equal)

[Function]

Returns 0FFFFH (true) when the 1st and 2nd term values are not equal or 0000H (false) when those values are equal.

A blank space is necessary before and after ‘NE’.

[Usage Example]

```
A1    EQU   0AH
A2    EQU   2H

      MOV   A, # (A1 NE A2)          AND 0FH ; (c)
      MOV   A, # (A1 NE (A2+8H))     AND 0FH ; (d)
```

In the case of (c),

“A1 NE A2” becomes “0AH NE 2H”.

Since the 1st and 2nd term values are not equal, 0FFFFH is returned, the logical product with 0FH is obtained and 0FH is loaded into the A register.

In the case of (d),

“A1 NE (A2 + 8H)” becomes “0AH NE (2H + 8H)”.

Since the 1st and 2nd term values are equal, 0000H is returned, the logical product with 0FH is obtained and 0H is loaded into the A register.

(c) GT or > (greater than)

[Function]

Returns 0FFFFH (true) when the 1st term value is greater than the 2nd term value or 0000H (false) when the 1st term value is equal to or less than the 2nd term value.

A blank space is necessary before and after ‘GT’.

[Usage Example]

```
A1    EQU   7H
A2    EQU   5H

      MOV   A, # (A1 GT A2)          AND 0FH ; (e)
      MOV   A, # (A1 GT (A2+2H))     AND 0FH ; (f)
```

In the case of (e),

“A1 GT A2” becomes “7H GT 5H”.

Since the 1st term value is greater than the 2nd term value, 0FFFFH is returned, the logical product with 0FH is obtained and 0FH is loaded into the A register.

In the case of (f),

“A1 GT (A2 + 2H)” becomes “7H GT (5H + 2H)”.

Since the 1st and 2nd term values are equal, 0000H is returned, the logical product with 0FH is obtained and 0H is loaded into the A register.

(d) GE or >= (greater or equal)

[Function]

Returns 0FFFFH (true) when the 1st term value is greater than or equal to the 2nd term value or 0000H (false) when the former is less than the latter.

A blank space is necessary before and after ‘GE’.

[Usage Example]

```
A1     EQU    8H
A2     EQU    3H

       MOV    A, # (A1 GE A2)          AND 0FH ; (g)
       MOV    A, # (A1 GE  (A2+6H))    AND 0FH ; (h)
```

In the case of (g),

“A1 GE A2” becomes “8H GE 3H”.

Since the 1st term value is greater than the 2nd term value, 0FFFFH is returned, the logical product with 0FH is obtained and 0FH is loaded into the A register.

In the case of (h),

“A1 GE (A2 + 6H)” becomes “8H GE (3H + 6H)”.

Since the 1st term value is less than the 2nd term, 0000H is returned, the logical product with 0FH is obtained and 0H is loaded into the A register.

(e) LT or < (less than)

[Function]

Returns 0FFFFH (true) when the 1st term value is less than the 2nd term value or 0000H (false) when the 1st term value is equal to or greater than the 2nd term value.

A blank space is necessary before and after ‘LT’.

[Usage Example]

```
A1    EQU   1H
A2    EQU   0AH

      MOV   A, # (A1 LT A2)          AND 0FH ; (i)
      MOV   A, # ((A1+0AH) LT A2)    AND 0FH ; (j)
```

In the case of (i),

“A1 LT A2” becomes “1H LT 0AH”.

Since the 1st term value is less than the 2nd term value, 0FFFFH is returned, the logical product with 0FH is obtained and 0FH is loaded into the A register.

In the case of (j),

“(A1 + 0AH) LT A2” becomes “1H + 0AH LT 0AH”.

Since the 1st term value is greater than the 2nd term value, 0000H is returned, the logical product with 0FH is obtained and 0H is loaded into the A register.

(f) LE or <= (less or equal)

[Function]

Returns 0FFFFH (true) when the 1st term value is less than or equal to the 2nd term value or 0000H (false) when the former is greater than the latter.

A blank space is necessary before and after ‘LE’.

[Usage Example]

```
A1    EQU   5H
A2    EQU   9H

      MOV   A, # (A1 LE A2)           AND 0FH ; (k)
      MOV   A, # ((A1+5H)   LE A2)    AND 0FH ; (l)
```

In the case of (k),

“A1 LE A2” becomes “5H LE 9H”.

Since the 1st term value is less than the 2nd term value, 0FFFFH is returned, the logical product with 0FH is obtained and 0FH is loaded into the A register.

In the case of (l),

“(A1 + 5H) LE A2” becomes “(5H + 5H) LE 9H”.

Since the 1st term value is greater than the 2nd term value, 0000H is returned, the logical product with 0FH is obtained and 0H is loaded into the A register.

(4) Shift Operators

(a) SHR (right shift)

[Function]

Shifts the 1st term value to the right by the value (No. of bits) indicated by the 2nd term value and returns the shifted value.

Zeros equal to the number of shifted bits are set in the most significant bits.

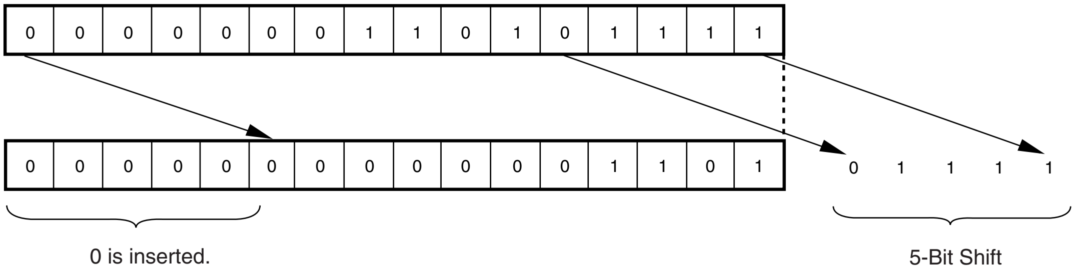
A blank space is necessary before and after ‘SHR’.

[Usage Example]

```
FIELD :     MOV     A, #1AFH     SHR 5        ;  (a)
FLAG :      MOV     H, #30H.1    SHR 6        ;  (b)
```

In the case of (a),

1AFH is shifted to the right by 5 bits.

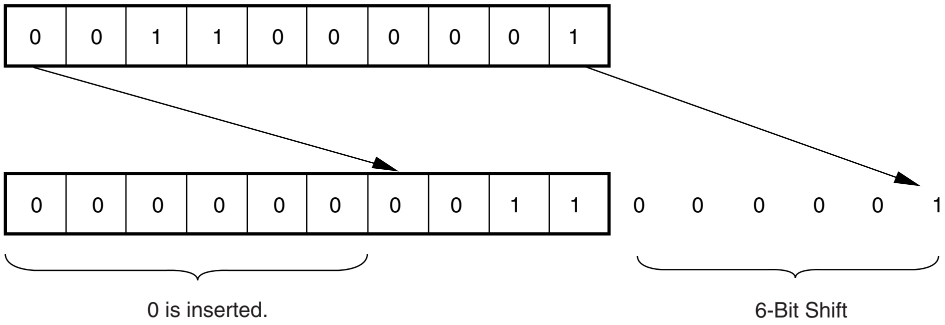


Accordingly, the value 0DH is loaded into the A register.

Thus, (a) can also be described as “MOV A, #0DH”.

In the case of (b),

30H.1 is shifted to the right by 6 bits.



Accordingly, the value 03H is loaded into the register.

Thus, (b) can also be described as “MOV H, #03H”.

(b) SHL (left shift)

[Function]

Shifts the 1st term value to the left by the value (No. of bits) indicated by the 2nd term value and returns the shifted value.

Zeros equal to the number of shifted bits are set in the least significant bits.

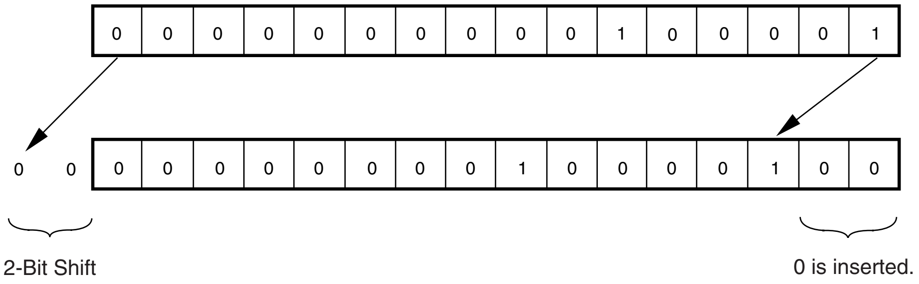
A blank space is necessary before and after ‘SHL’.

[Usage Example]

```
FLY :    MOV    XA, #21H     SHL 2              ; (c)
LSB2 :   MOV    XA, #0BFH  SHR 2  SHL 2         ; (d)
```

In the case of (c),

21H is shifted to the left by 2 bits.

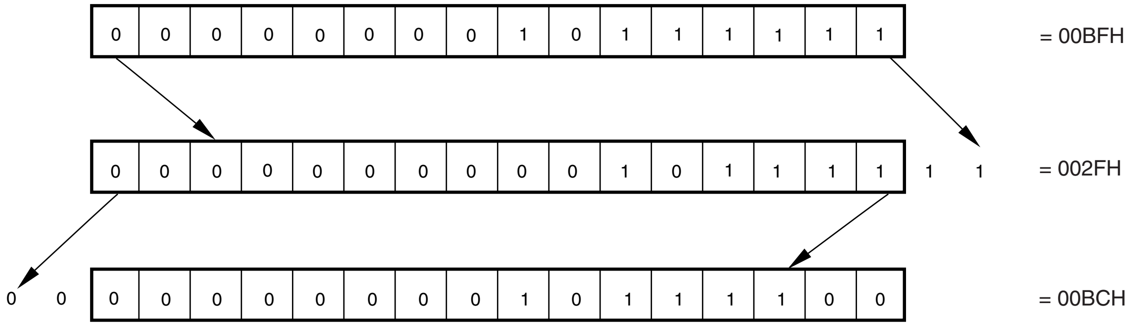


Accordingly, the value 84H is returned.

Thus, (c) can also be described as “MOV XA, #84H”.

In the case (d),

0BFH is shifted to the right by 2 bits and to the left by 2 bits.



```
        0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1        = 00BFH
        0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1  1 1   = 002FH
    0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0        = 00BCH
```

Accordingly, 0BCH is loaded into the XA register.

Thus, (d) can also be described as “MOV XA, #0BCH”.

This has been obtained by shifting the address to the right by 2 bits and then shifting it to the left by 2 bits. Since the left bit shift sets the least significant 2 bits to 0, the operation is the same as the masking of the least significant 2 bits.

Thus, (d) can also be described as follows:

```
LBS2 :    MOV    XA, #0BFH   AND    0FCH
```

(5) Bit Location Specification Operator

(a) . (bit location specification)

[Function]

Calculates the bit address from the 1st and 2nd term bit locations and returns the calculated bit address.

[Description]

The 1st and 2nd terms have restrictions.

<1> 1st term restrictions

The 1st term must be in the range from 000H to 0FFFH.
The formula attribute is NUMBER term or absolute DATA term.
Refer to **Table 3-12. Symbol Attribute Types for Operation**.

<2> 2nd term restrictions

When the symbol is described in the 2nd term, the symbol attribute is NUMBER only. The 2nd term value is an absolute value from 0 to 3.

[Usage Example]

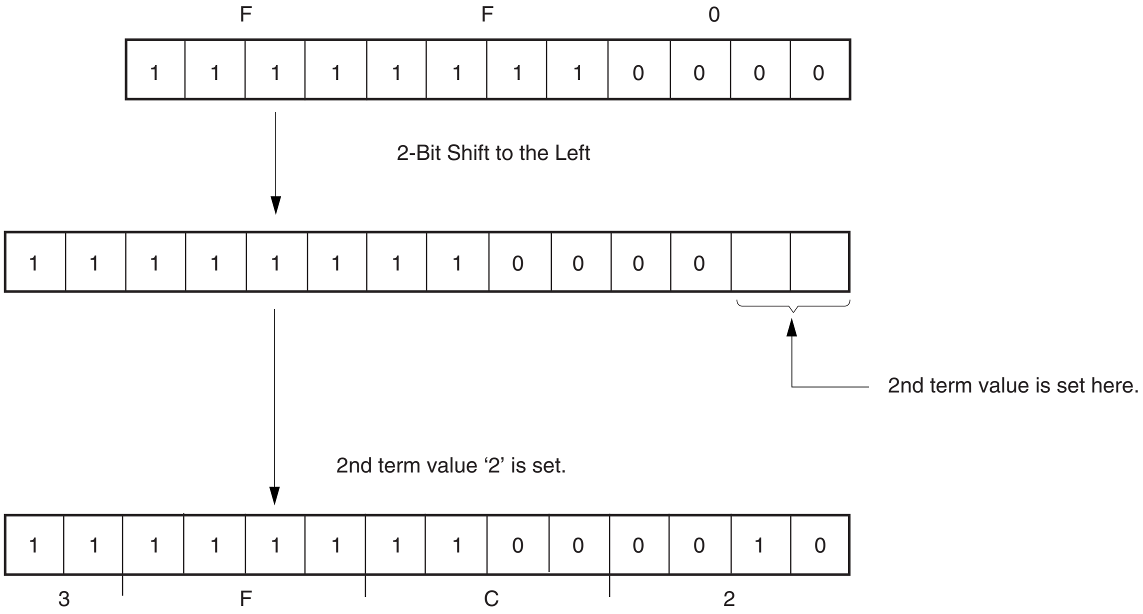
```
SYM :       EQU     PORT0.2        ; (a)
BIT1 :      CLR1    36H.1+1        ; (b)
BIT2 :      CLR1    36H.1+1H.0     ; (c)
```

In the case of (a),

PORT0 is a reserved word having the value 0FF0H.

Symbol ‘SYM’ has the value of 0FF0H × 4 + 2 = 3FC2H.

The bit address is obtained by shifting the 1st term value to the left by 2 bits and setting the 2nd term value to the empty least significant 2 bits.



Thus, the bit address becomes 3FC2H and symbol ‘SYM’ has the value 3FC2H.

In the case of (b),

The assembler internally possesses 36H.1 as the following type data.

```
  0   0   1   1   0   1   1   0   0   1     = 36H.1 (Bit Address 0D9H)
```

1 is added to this value.

```
  0   0   1   1   0   1   1   0   0   1
+                                     1
--------------------------------------------
  0   0   1   1   0   1   1   0   1   0     = 36H.2 (Bit Address 0DAH)
```

Thus, 36H.1 + 1 = 36H.2.

Note that the result is not 36H.1 + 1 = 37H.1.

In the case of (c),

36H.1 + 1H.0 becomes as follows:

```
  0   0   1   1   0   1   1   0   0   1     = 36H.1 (bit address 0D9H)
+                 0   0   0   1   0   0     = 1H.0 (bit address 4H)
--------------------------------------------
  0   0   1   1   0   1   1   1   0   1     = 37H.1 (bit address 0DDH)
```

Thus, 36H.1 + 1H.0 = 37H.1

(6) Byte Separation Operators

(a) HIGH

[Function]

Returns the high-order 8 bits of the term.
There must be a space between HIGH and the term.

[Usage Example]

```
            ORG     1234H
START :       .
              :

            MOV     A, #HIGH START          ; <1>
```

[Description]

As the label ‘START’ has a value of 1234H, the value of the high-order 8 bits, 12H, is returned.
Therefore, <1> can also be written as “MOV A, #12H”.

(b) LOW

[Function]

Returns the low-order 8 bits of the term.
There must be a space between LOW and the term.

[Usage Example]

```
            ORG     5678H
WORK :        .
              :

            MOV     A, #LOW WORK            ; <2>
```

[Description]

As the label ‘WORK’ has a value of 5678H, the value of the low-order 8 bits, 78H, is returned.
Therefore, <2> can also be written as “MOV A, #78H”.

(7) Other Operator

(a) ( )

[Function]

Calculation in ( ) is carried out ahead of calculation outside ( ).

This function is used to change the operation priority order.

[Usage Example]

```
MOV     A, # (1+3)        *2
```

$$\underbrace{\underbrace{(1+3)}_{<1>}\ *2}_{<2>}$$

Operation is carried out in the order of <1> and <2> and 8H is returned.

If there is no ( ),

$$\underbrace{1+\underbrace{3\ *2}_{<1>}}_{<2>}$$

Operation is carried out in the order or <1> and <2> and 7H is returned.

Refer to **Table 3-8** for details of the operator priority order.

3.4.2 Operation Restrictions

Formula calculation is carried out with terms combined using operators. Constants, $, names and labels can be described as terms and each term has relocation and symbol attributes.

Possible operators for the particular terms are limited depending on the relocation and symbol attribute types of those terms. Thus, when describing a formula, it is important to take note of the relocation and symbol attributes of the terms forming the formula.

(1) Operation and relocation attributes

Each term forming the formula has relocation attributes.

The relocation attribute types, characteristics and the corresponding terms are shown below.

Table 3-9. Relocation Attribute Types

Type	Characteristics	Corresponding Term
Absolute term	Values to be determined upon assembly	• Constant • Label defined in the absolute segment and its segment name • $ indicating the location address defined in the absolute segment • Constant, the above label and the name which defines the above $
Relocatable term	Terms not to be determined upon assembly	• Label defined in the relocatable segment and its segment name • $ indicating the location address defined in the relocatable segment • Name which defined the relocatable label
External reference term	Term for external reference to another module symbol	• Label defined by EXTRN pseudo-instruction • Reserved word STACK (the assembler automatically defines STACK as an external reference symbol)

Permissible combinations of the operators and terms are classified by relocation attributes in the **table below (except external reference terms)**.

Table 3-10. Combination of Terms and Operators Classified by Relocation Attributes (Except External Reference Terms)

Relocation Attributes of Term Operator Type	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	A	A	R	R
– X	A	A	—	—
X MOD Y	A	—	—	—
NOT X	A	A	—	—
X AND Y	A	—	—	—
X OR Y	A	—	—	—
X XOR Y	A	—	—	—
X EQ Y	A	—	—	A[Note]
X NE Y	A	—	—	A[Note]
X GT Y	A	—	—	A[Note]
X GE Y	A	—	—	A[Note]
X LT Y	A	—	—	A[Note]
X LE Y	A	—	—	A[Note]
X SHL Y	A	—	—	—
X SHR Y	A	—	—	—
X . Y	A	—	—	—
HIGH X	A	A	R	R
LOW X	A	A	R	R

Note Operation enable only between the symbols defined in the same segment.

Caution If the term is a relocatable term or external reference term, nesting is not possible. If used in combination with a BRCB, EQU or SET instruction, only an absolute term can be used.

Remark

• ABS	:	Absolute term
• REL	:	Relocatable term
• A	:	Operation result is the absolute term.
• R	:	Operation result is the relocatable term.
• —	:	Operation impossible

Four operators can operate the external reference terms. They are '+', '–', '.', and 'NOT'.

Executable combinations of these operators and external reference terms are classified by relocation attributes in the **table below (except reference terms)**.

Table 3-11. Combinations of Terms and Operators Classified by Relocation Attributes (External Reference Terms)

Relocation Attributes of Term	X : ABS	X : EXT	X : REL	X : EXT	X : EXT
Operator Type	Y : EXT	Y : ABS	Y : EXT	Y : REL	Y : EXT
X + Y	E	E	—	—	—
X – Y	—	E	—	—	—
+ X	A	E	R	E	E
– X	A	—	—	—	—
NOT X	A	—	—	—	—
X . Y	—	E	—	E	—
HIGH X	A	E	R	E	E
LOW X	A	E	R	E	E

Caution If the term is a relocatable term or external reference term, nesting is not possible. If used in combination with a BRCB, EQU or SET instruction, only an absolute term can be used.

Remark
- ABS : Absolute term
- REL : Relocatable term
- EXT : External reference term
- A : Operation result is the absolute term.
- R : Operation result is the relocatable term.
- E : Operation result is the external reference term.
- — : Operation impossible

Caution

Only '+', '–', '.', and 'NOT' operators can be used for the external reference name.

Thus, the following description cannot be made.

Module 1

```
EXTRN BIT (FLAG1)

MOV    H,#FLAG1        SHR 6
CLR1   CY
OR1    CY, @H+FLAG1
         ≀
```

Module 2

```
          PUBLIC          FLAG1

FLAG1  EQU    30H.3
```

Operation (SHR) cannot be carried out for the external reference name FLAG1.
Thus, this description results in an error.

Since the above description is an error, avoid it as follows:

Module 1

```
EXTRN BIT (FLAG1)
EXTRN FLAG                ... <1>

MOV    HL,#FLAG           ... <2>

CLRI   CY

ORI    CY, @H+FLAG1
```

Module 2

```
          PUBLIC FLAG, FLAG1

FLAG   EQU    30H
FLAG1  EQU    30H.3
```

Refer to the symbol indicating FLAG1 data address (30H) in <1> and set the most significant 4-bit address of FLAG1 to the H register in <2> using the symbol.

All operators can be used for the absolute symbol.

(2) Operation and symbol attributes

Each term forming the formula has symbol attributes in addition to the relocation attributes.
The symbol attribute types for operation and the corresponding terms are shown in the table below.

Table 3-12. Symbol Attribute Types for Operation

Symbol Attribute Types	Corresponding Terms
NUMBER term	• Constant • Constant symbol (having symbol attribute ‘NUMBER’)
CODE term	• Code symbol (having symbol attribute ‘CODE’) • ‘$’ defined in the code segment
DATA term	• Data symbol (having symbol attribute ‘DATA’) • ‘$’ defined in the data segment
BIT term	• Bit symbol (having symbol attribute ‘BIT’) • Bit operator using constant (bit value)
PBIT term	• Port bit symbol (having symbol attribute ‘PBIT’)

Combinations of operation possible operators and terms are classified by symbol attributes as shown in **Table 3-13**.

Table 3-13. Combinations of Terms and Operators Classified by Symbol Attributes

(1) Binary operators

Attributes	X : Num	X : Code	X : Data	X : Bit	X : Num	X : Num	X : Num	X : Code	X : Data	X : Bit
Operators	Y : Num	Y : Num	Y : Num	Y : Num	Y : Code	Y : Data	Y : Bit	Y : Code	Y : Data	Y : Bit
Operator X + Y	N	C	D	B	—	D	B	—	—	B
X – Y	N	C	D	B	—	—	—	N[Note 1]	N[Note 1]	N[Note 1]
X * Y	N	N	N	N	—	—	—	—	—	—
X / Y	N	N	N	N	—	—	—	—	—	—
X MOD Y	N	N	N	N	—	—	—	—	—	—
X AND Y	N	N	N	N	N	N	N	—	—	—
X OR Y	N	N	N	N	N	N	N	—	—	—
X XOR Y	N	N	N	N	N	N	N	—	—	—
X SHL Y	N	N	N	N	—	—	—	—	—	—
X SHR Y	N	N	N	N	—	—	—	—	—	—
X EQ Y	N	N	N	N	N	N	N	N	N	N
X NE Y	N	N	N	N	N	N	N	N	N	N
X LT Y	N	N	N	N	N	N	N	N	N	N
X LE Y	N	N	N	N	N	N	N	N	N	N
X GT Y	N	N	N	N	N	N	N	N	N	N
X GE Y	N	N	N	N	N	N	N	N	N	N
X . Y	B	—	B[Note 2]	—	—	—	—	—	—	—
HIGH X	N	C	D	—	N	N	N	C	D	—
LOW X	N	C	D	—	N	N	N	C	D	—

(2) Unary operators

Attributes of X	Num	Code	Data	Bit	Pbit
Operator NOT X	N	—	—	—	—
+ X	N	C	D	B	—
– X	N	—	—	—	—

Notes 1. These operations are only possible when the 1st and 2nd terms of the formula (X and Y in the table) are defined in the same segment. Otherwise, errors will result.

2. This is the case when a bit operator is used for the specific address name code. When the value is FB0H.0 to FBFH.3 or FF0H.0 to FFFH.3, the operation result is the PBIT attribute.

★ **Remark**

Num	:	NUMBER attribute	Code	:	CODE attribute
Data	:	DATA attribute	Bit	:	BIT attribute
Pbit	:	PBIT attribute			
N	:	The operation result is the NUMBER attribute.			
C	:	The operation result is the CODE attribute.			
D	:	The operation result is the DATA attribute.			
B	:	The operation result is the BIT attribute.			
P	:	The operation result is the PBIT attribute.			

Caution

Only the bit operator can be used for the specific address name code. If a specific address name code is included in other formulas, errors will result.

Example

PORT0 + 1	:	Error
PORT0.1	:	Operation possible
PORT0.1 + 1	:	Error

3.5 OPERAND CHARACTERISTICS

Instructions requiring operands (instructions and pseudo-instructions) have different operand value sizes, ranges and symbol attributes depending on their types.

3.5.1 Symbol Addressing

If a symbol is described in the operand column, the address or value assigned for the symbol is interpreted as the operand value.

```
HERE :    BR     !THERE ...... Branches to the address assigned for ‘THERE’.
           :
           :
THERE :   SET1   0FH.1
VALUE     EQU    10H
          MOV    A,VALUE ...... Has the same meaning as “MOV A,10H”.
```

When the symbol is referred to, the assembler checks the symbol attribute and its value. If the symbol attribute or symbol value is not appropriate as an operand for the instruction, it results in an error.

When the symbol is referred to as an instruction operand, the following symbol attributes and their values can be referred to.

(1) Program memory (ROM) addressing

If a symbol is used for program memory addressing, checking described in **Table 3-14. Symbol Attributes Enabled for Reference (1)** is carried out. In addition, whether the symbol value is in the range of ROM incorporated into the assembled product type specified by -C option. If the symbol value is outside the ROM range, an error results.

Table 3-14. Symbol Attributes Enabled for Reference (1)

Identifier (Value Range)	Symbol Attribute Enabled for Reference				
	NUMBER	CODE	DATA	BIT	PBIT
addr1 (Whole ROM range)	○	○	—	—	—
addr (0H to 3FFFFH)	○	○	—	—	—
caddr (In the same block)	○	○	—	—	—
faddr (0H to 7FFH)	○	○	—	—	—
taddr (Even values in 20H to 7FH)	○	○	—	—	—

Remark ○ : Reference possible
— : Reference not possible (reference results in errors)

(2) Data memory (RAM) addressing

If a symbol is used for data memory addressing, checking described in **Table 3-15. Symbol Attributes Enabled for Reference (2)** is carried out. In addition, whether the symbol value is included in the data memory incorporated into the assembled product type specified by -C option or the I/O in the RAM addresses 0F80H to 0FFFH. If the symbol value is not included in them, an error results.

As for addressing operations marked with circles in the column of “specific address name code R/W attribute check” in **Table 3-15**, if the specific address name code defined in the range from 0F80H to 0FFFH is used as an operand, whether the access is READ or WRITE access enabled for the specific address name code is also checked.

Table 3-15. Symbol Attributes Enabled for Reference (2)

Identifier (Value Range)	Symbol Attribute Enabled for Reference					Specific Address Name Code R/W Attribute Check
	NUMBER	CODE	DATA	BIT	PBIT	
mem (Whole RAM range)	○	—	○	—	—	○
pmem (FC0H to FFFH)	○	—	○	—	—	—
mem.bit (0H.0 to FFFH.3)	—	—	—	○	○	○
@H + mem.bit (Whole RAM range)	—	—	—	○	○	—
fmem.bit (FB0H.0 to FBFH.3, FF0H.0 to FFFH.3)	—	—	—	—	○	○

Remark ○ : Reference possible
— : Reference not possible (reference results in errors)

Caution

1. When the operand identifier is mem, whether the symbol value is in the RAM range is checked. After that, only the hexadecimal lower 2 digits are padded into the object and the higher 1 digit (100H for the symbol value 130H, for example) is not padded.
2. When carrying out memory bit manipulation, the object code mem.bit or fmem.bit is generated from the symbol described in the operand column. Object code mem.bit is normally generated.

   However, if the symbol is a reserved word in the 0FB0H.0 to 0FBFH.3 or 0FF0H.0 to 0FFFH.3 range, object code fmem.bit is generated.

   To generate the fmem.bit object code, a reserved word in the above range must be specified. If the reserved word is in the above range, the mem.bit object code is generated when immediate data is specified.

   Examples are shown below.

Example 1. SET1 PORT0.1

PORT0 is a reserved word mapped at address 0FF0H.
Thus, the fmem.bit object code is generated.

2. SET1 0FF0H.1

Although 0FF0H is in the range enabling fmem.bit to be generated, the mem.bit object code is generated because immediate data has been specified.

★ 3. If a specific address name code is used when register indirect addressing (@H + mem.bit,pmem.@L) is included in the operand, READ and WRITE access attributes are not checked.

(3) Immediate data

Symbol attributes enabled as DB pseudo-instruction operands are the same as with n8 (0H to FFH) in the table below.

Table 3-16. Symbol Attributes Enabled for Reference (3)

Identifier (Value Range)	Symbol Attribute Enabled for Reference (3)				
	NUMBER	CODE	DATA	BIT	PBIT
n4 (0H to FH)	○	△	△	△	△
n8 (0H to FFH)	○	△	△	△	△

Remark ○ : Reference possible
△ : Reference possible but range check not executed.
(Assembled using the least significant 4 or 8 bits of the symbol value)

3.5.2 Operand Value Size and Range

In the case of an instruction, the numeric value or name describable as an operand, the label value size and address range are determined by the operand identifier of the instruction set.

For details, refer to the User's Manual for each device.

CHAPTER 4 PSEUDO-INSTRUCTIONS

In this chapter, the types and functions of pseudo-instructions which describe in the source program will be described.

★ For details of linker option and assembler option (-Xoption, -XXoption, and -XXXoption), refer to the **Operation** Manual.

4.1 OUTLINE OF PSEUDO-INSTRUCTIONS

Pseudo-instructions are described in the source program as is the case with instructions. They are used to provide various instructions when assembler package carries out a series of operations.

Instructions are converted into object codes (machine codes) as a result of assembly. However, pseudo-instructions are not converted into object codes in principle.

Pseudo-instructions have the following functions:

- Facilitate source program description
- Execute memory initialization and area reserve
- Provide the assembler and linker with the necessary information for processing operations.

Table 4-1 gives a listing of pseudo-instruction types.

Table 4-1. List of Pseudo-Instruction Types

Pseudo-Instruction Type	Pseudo-Instruction
Segment definition pseudo-instruction	CSEG, DSEG, ORG
Program linkage pseudo-instruction	NAME, PUBLIC, EXTRN
Symbol definition pseudo-instruction	EQU, SET
Data definition pseudo-instruction	DB
Area reserve pseudo-instruction	DS, STKLN
Branch instruction auto select pseudo-instruction	BR
Vector entry table definition pseudo-instruction	VENTn
GETI instruction table definition pseudo-instruction	TCALL, TBR
Assembly end pseudo-instruction	END

Each pseudo-instruction is described in detail below.

In the following description, brackets mean that items in them can be omitted and ... means the repetition of the same format. For example, when [(size)] [initial value [, ...]] is described, the following description is possible.

- (Size)
- (Size) initial value 1, initial value 2, initial value 3
- Initial value 1, initial value 2

4.2 SEGMENT DEFINITION PSEUDO-INSTRUCTIONS

The segment is a block of the same type routines or data and the segment definition pseudo-instruction is a pseudo-instruction to declare the segment start.

The following four types of segments are available:

- Code segment
- Data segment
- Absolute segment
- Stack segment

The type of segment determines in which range of the memory the address is located.
Each segment definition procedure and the located memory address are shown in the table below.

Table 4-2. Segment Definition Procedure and Memory Address to be Located

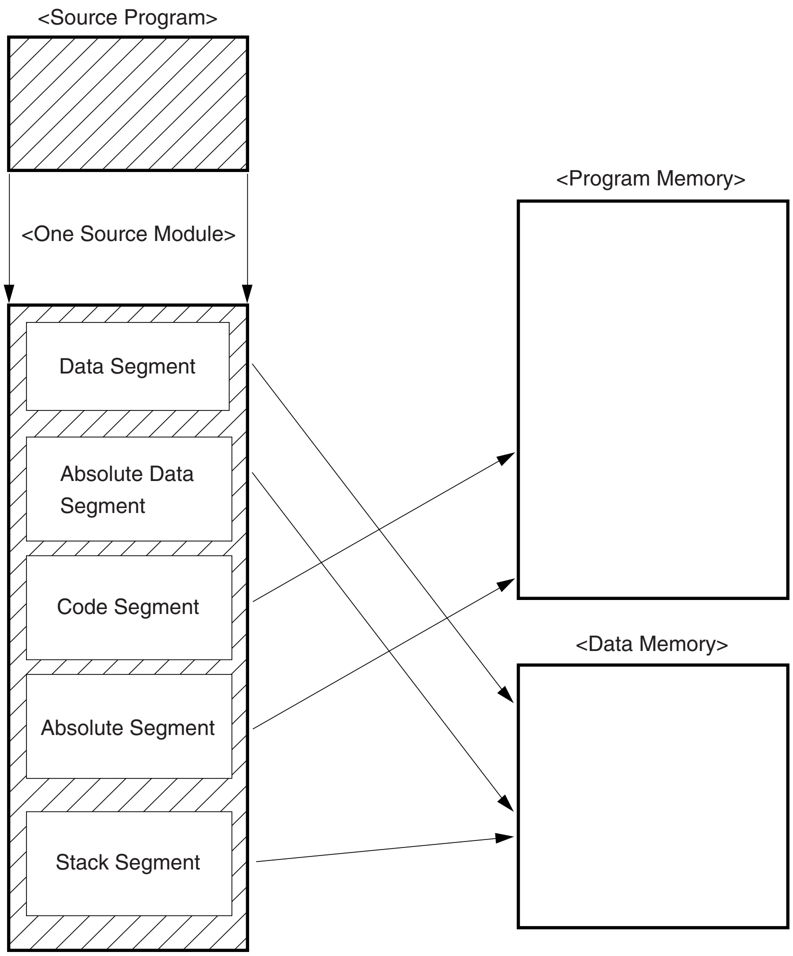
Segment Type	Definition Procedure	Memory Address to be Located
Code segment	CSEG pseudo-instruction	In program memory (ROM)
Data segment	DSEG pseudo-instruction	In data memory (RAM)
Absolute segment	ORG pseudo-instruction	In program memory (ROM)
Stack segment	Generated only when STKLN pseudo-instruction is specified.	In data memory (RAM)

When the user wants to determine the memory location address, describe the absolute segment.
A segment location example is shown in **Figure 4-1. Segment Definition and Memory Location**.

Caution

- Until the first segment definition pseudo-instruction is generated in the source program (even if the segment definition pseudo-instruction has not been described anywhere in the source program), the segment is interpreted as an absolute code segment with the start address set to 10H.
- When using a list converter, describe the segment definition pseudo-instruction before describing an object code generating instruction.

Figure 4-1. Segment Definition and Memory Location



(1) CSEG (code segment)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
Segment name	CSEG	(Relocation attribute)	[; comment]

[Functions]

- CSEG pseudo-instruction instructs the assembler to start the code segment.
- Instructions which are described after this pseudo-instruction until the segment definition pseudo-instruction (CSEG, DSEG) appears again will belong to the code segment. They will be located in the ROM address upon final conversion into machine words.
- The code segment location address range can be further limited by specifying the relocation attribute in the operand column of CSEG pseudo-instruction.

  There are eight relocate instructions listed in **Table 4-3**.

[Functions are listed in **Table 4-3** and the relations between attributes and program memory are shown in **Figure 4-2**.]

Table 4-3. Relocation Attribute Functions

Item	Relocation Attribute	Function
1	INBLOCKA	Specifies that the code segment be relocated in any one block in the program memory ("block" means the 4-Kbyte area of X000H to XFFFH). For the segment including BRCB instruction or BR pseudo-instruction for reference to its own segment, this attribute or INBLOCK attribute is specified.
2	XBLOCKA	Specifies that the code segment be relocated at any position in the program memory (the segment of this attribute may be located over more than one block).
3	INBLOCK	Specifies that the code segment be relocated in any one block in the 0000H to 3FFFH range of the program memory. For the segment including BRCB instruction or BR pseudo-instruction for reference to its own segment, this attribute or INBLOCKA attribute is specified. For the segment referred to by "BR !addr (branch instruction to 0H to 3FFFH)" or "CALL !addr (branch instruction to 0H to 3FFFH or call instruction)", this attribute or XBLOCK attribute is specified.
4	XBLOCK	Specifies that the code segment be relocated in any location in the 0000H to 3FFFH range of the program memory (the segment of this attribute may be located over more than one block). For the segment referred to by "BR !addr (branch instruction to 0H to 3FFFH)" or "CALL !addr (branch instruction to 0H to 3FFFH or call instruction)", this attribute or INBLOCK attribute is specified.
5	SENT	Specifies that the code segment be relocated in the 0000H to 07FFH range of the program memory. This attribute is specified for the segment having the internal entry address referred to by the 2-byte subroutine call instruction "CALLF !faddr".
6	IENT	Specifies that the code segment be relocated in the 0020H to 007FH range of the program memory. The start address of this segment is an even address. This attribute is specified for the segment including the GETI instruction table.
7	PAGE	Specifies that the start address of the code segment be relocated on the 256-byte boundary (XX00H) in the program memory. This attribute can be used in combination with INBLOCKA, XBLOCKA, INBLOCK, XBLOCK and SENT attributes (if only PAGE attribute is specified, "INBLOCK PAGE" is regarded as having been specified). This attribute is specified for the segment including the table for table reference instruction MOVT PCXA and MOVT PCDE or inter-register indirect branch instructions BR PCDE and BR PCXA.
8	AT absolute formula	Specifies that the code segment be relocated at the formula absolute address specified using 'AT absolute formula'.

CSEG **code segment** **CSEG**

Next, the rules for relocation attribute selection are described.

- Segment for relocation address specification by absolute address .... AT absolute formula attribute
- Segment including GETI instruction table .... IENT attribute
- Segment referred to by CALLF !faddr instruction .... SENT attribute

- Segments with 4 Kbytes or less including BRCB instruction and BR pseudo-instruction for own-segment reference
  - Segments externally referred to by “BR !addr”Note 1 or “CALL !addr”Note 2 .... INBLOCK attribute
  - Segment for relocation in the 0000H to 3FFFH range .... INBLOCK attribute
  - Other segment .... INBLOCKA attribute

  * When locating on the 256-byte boundary, use PAGE attribute as well.

- All other segments
  - Segment externally referred to by “BR !addr”Note 1 or “CALL !addr”Note 2 .... XBLOCK attribute
  - Segment for relocation in the 0000H to 3FFFH range .... XBLOCK attribute
  - Other segments .... XBLOCKA attribute

  * When location on the 256-byte boundary, use PAGE attribute as well.

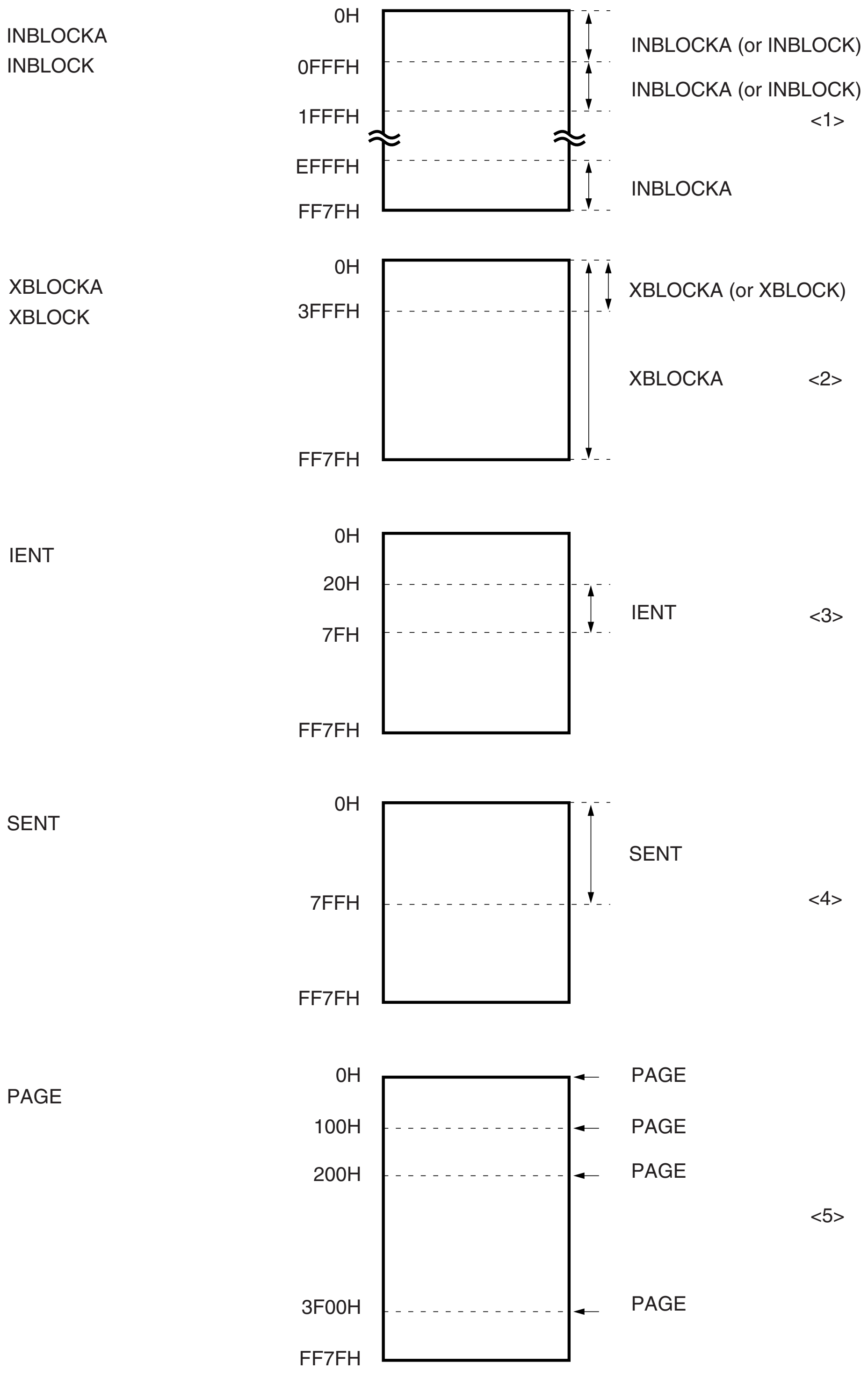
Notes 1. “BR !addr” is a branch instruction for the 0000H to 3FFFH (16 Kbytes) range.
2. “CALL !addr” is a call instruction for the 0000H to 3FFFH (16 Kbytes) range.

Caution

INBLOCK and XBLOCK, INBLOCKA and XBLOCKA are relocation attributes having similar meaning. The summary of their differences follows:

	Relocatable Range	**Relocation Block**
INBLOCK	000H to 3FFFH	Each segment with INBLOCK or INBLOCK specified is relocated in one block. Thus, the maximum size of each segment becomes equal to the block size (4 Kbytes). If there are several small segments having more than one segment, two or more segments may be relocated in one block.
INBLOCKA	Whole program memory (with a maximum of 64 Kbytes)	
XBLOCK	000H to 3FFFH	Each segment with XBLOCK or XBLOCKA specified is relocated irrespective of the block. Thus, the maximum size of each segment is not affected by the block size (4 Kbytes).
XBLOCKA	Whole program memory (with a maximum of 64 Kbytes)	

Figure 4-2. Relocation Attributes and Program Memory



CSEG **code segment** **CSEG**

[Applications]

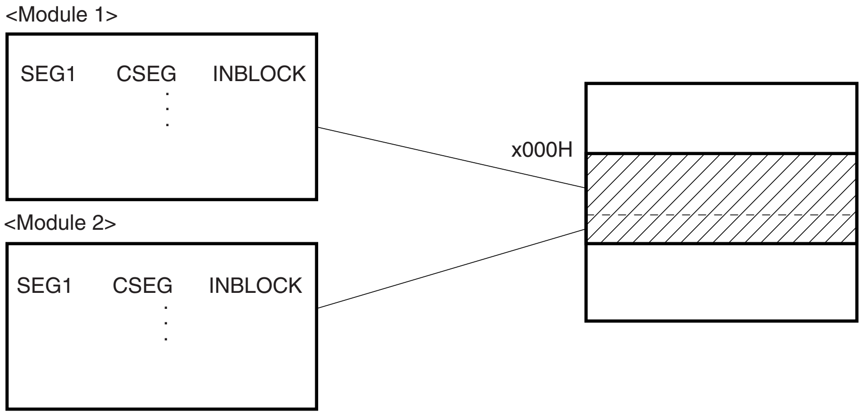
- Describe an instruction or DB pseudo-instruction for the code segment defined by CSEG pseudo-instruction.
- Refer to **Table 4-3. Relocation Attribute Functions** for details concerning the relocation attribute operating procedure.
- Define the single-function unit including a subroutine as one code segment. If the unit has a relatively large size or the subroutine can be put to widespread use (including the development of another program), it is recommended to define the unit as one module.

[Description]

- Be sure to describe the segment name in the CSEG pseudo-instruction symbol column. If segment name description is omitted, an error results.
- A segment name can be referred to as symbol. In this case, the segment start address is used.
- CSEG pseudo-instruction remains valid until the next segment definition pseudo-instruction or END pseudo-instruction appears.
- The segment name automatically becomes the external definition symbol. In other modules, the segment name can be referred to using EXTRN declaration. In this case, PUBLIC declaration is not necessary. If PUBLIC declaration is described, an error results.

  For details of EXTRN and PUBLIC pseudo-instructions, refer to **4.3 PROGRAM LINKAGE PSEUDO-INSTRUCTIONS**.
- If relocation attributes are omitted, INBLOCK attribute is interpreted as having been specified.
- Segments having the same name are called the same name segments. If there is a same name segment in the module to be linked, the following processing operations are carried out.

(1) If the same name segments are in two or more different source modules and the relocation attributes of those segments are the same, the linker combines them into one segment.
The segments are linked in the same order as the object module order specified upon linkage.
When the segment name is referred to as the label, the symbol value becomes the start address of each segment before it is linked.



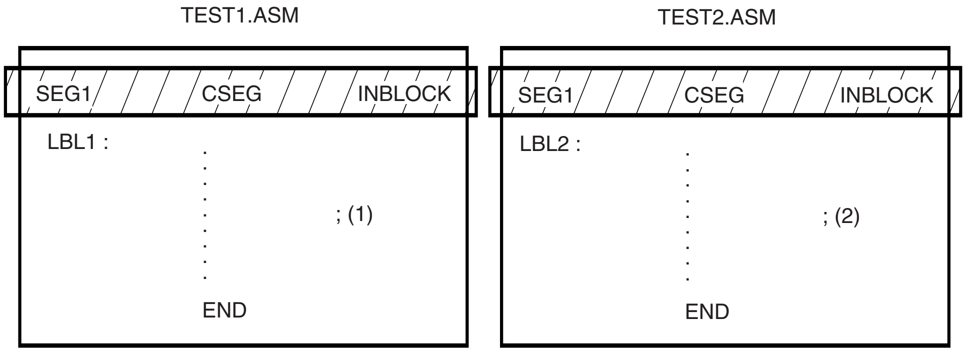
CSEG code segment CSEG

(2) If more than one same name segment has a different relocation attribute, an error results upon linkage.

(3) If the same name segments are in one source module, the assembler carries out operations (1) and (2) above.

- If no relocation attribute has been specified for the 2nd same name segment onward, the default INBLOCK attribute will not result. Instead, the same name segments will be regarded as having the previous relocation attributes and linked.
- If, when the assembler links homonymous the segment name is referred to as label, the symbol value will be the first address of the linked segments.

Example 1. If the same name segment having the same relocation attribute is different modules



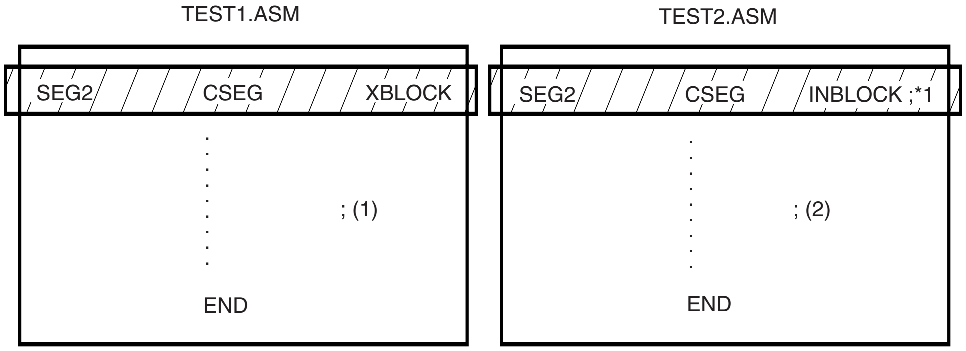
The same name code segment SEG1 is in each of two source modules TEST1.ASM and TEST2.ASM. Since these segments have the relocation attribute specified for the same INBLOCK attribute, they are linked upon linkage into one segment. As they are linked in the same order as the input module order specified upon linkage.

<a> When A > LK75X TEST1 TEST2 ⏎ is input, the segments are linked in the order of (1) and (2) starting with the smallest address.

<b> When A > LK75X TEST2 TEST1 ⏎ is input, the segments are linked in the order of (2) and (1).

If the segment name SEG1 is used as a symbol,
the same name as LBL1 in (1)
the same name as LBL2 in (2)
In the segment other than (1) and (2),
the same value as LBL1 in <a>
the same value as LBL2 in <b>
(If two or more same name segments are in the same source module, they will become the start address of the first segment name.)

Example 2. If the same name segment having a different relocation attribute is in different modules



TEST1.ASM

```
SEG2      CSEG      XBLOCK
          .
          .
          .
          .                ; (1)
          .
          .
          .
          END
```

TEST2.ASM

```
SEG2      CSEG      INBLOCK ;*1
          .
          .
          .
          .                ; (2)
          .
          .
          .
          END
```

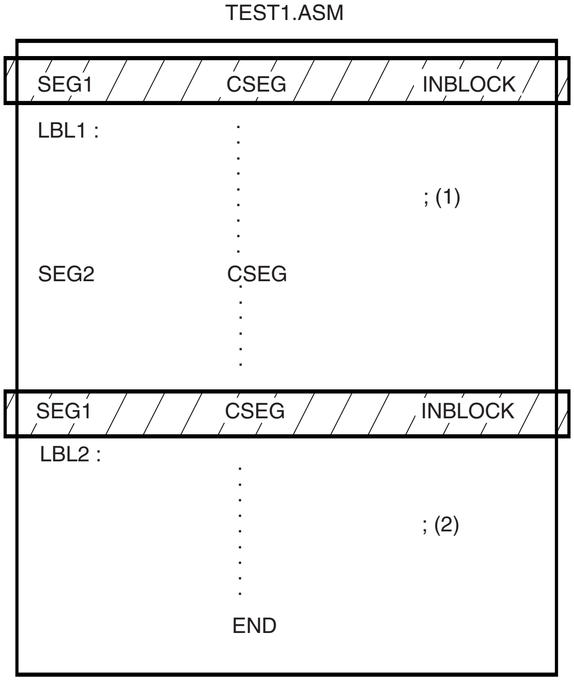
As in example 1, the same name code segment SEG2 is in each of two source modules TEST1.ASM and TEST2.ASM. Since these segments have the different relocation attributes (1) and (2) unlike in example 1, they will not become one segment. If linking the TEST1 object module with the TEST2 object module is attempted, an error will result.

If *1 statement is set as follows so that the relocate attribute is not specified in the first statement of segment (2) in TEST2.ASM,

```
SEG2      CSEG      ; *1
```

an error will also result. This is because, if no relocation attribute is specified for the segment pseudo-instruction, INBLOCK attribute will be provided as the default value and the segment will have the different relocation attribute from segment (1) (in this case, operation becomes different if TEST1.ASM and TEST2.ASM are loaded into one source module. (Refer to example 5)).

Example 3. When the same name segment having the same relocation attribute is in one module

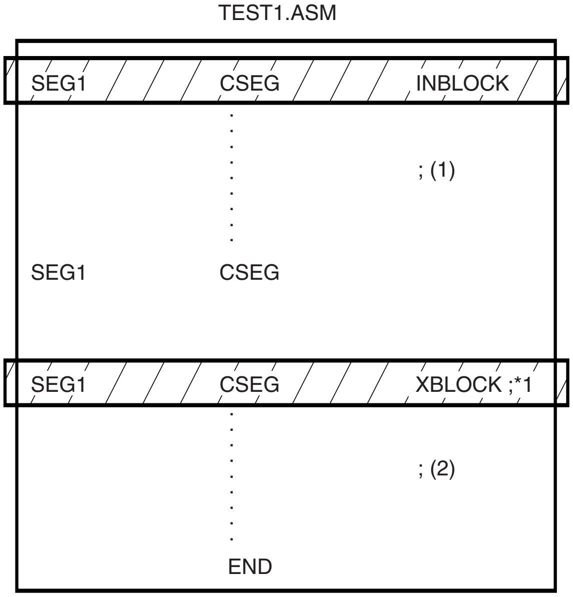


```
                       TEST1.ASM
SEG1                CSEG                INBLOCK
LBL1 :                .
                      .
                      .                 ; (1)
                      .
SEG2                CSEG
                      .
                      .
SEG1                CSEG                INBLOCK
LBL2 :                .
                      .
                      .                 ; (2)
                      .
                     END
```

The same name segment SEG1 having the same relocation attribute INBLOCK is in the source module TEST1.ASM. In this case, segment (2) is linked in the rear of segment (1) upon assembly and they are processed as one segment.

If the segment name SEG1 is referred to as a symbol, the start address (the same value as LBL1) of the linked segment will become the symbol value irrespective of reference locations.

4. When the same name segments having different relocation attributes in one module

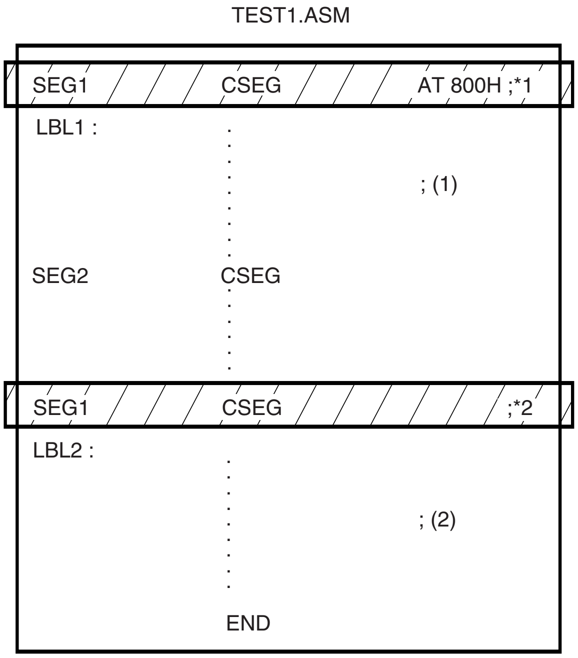


```
                       TEST1.ASM
SEG1                CSEG                INBLOCK
                      .
                      .
                      .                 ; (1)
                      .
SEG1                CSEG

SEG1                CSEG                XBLOCK ;*1
                      .
                      .
                      .                 ; (2)
                      .
                     END
```

Two same name segments SEG1 having different relocation attributes are in the source module TEST1.ASM. In this case, (1) and (2) are not linked because the relocation attributes are different. Statement *1 which defines segment (2) upon assembly becomes an error.

Example 5. When the same name segment with relocation attribute definition omitted is included in one module



TEST1.ASM

```
SEG1          CSEG          AT 800H ;*1
LBL1 :
                :
                :           ; (1)
                :
SEG2          CSEG
                :
                :
SEG1          CSEG                  ;*2
LBL2 :
                :
                :           ; (2)
                :
              END
```

Two same name segments (1) and (2) are in the source module TEST1.ASM. Although relocation attribute has been specified for segment (1), no specification has been made for segment (2) which was defined after (1). If no relocation attribute has been specified for the 2nd segment onward in one source module, the 2nd and succeeding segments are regarded as continuing from the 1st segment and are linked upon assembly. In this example, segment (1) and (2) are linked. Thus, after segment (1) is assembled as an absolute segment from the ROM address 800H, segment (2) is assembled as continuing from segment (1) and having an absolute attribute.

If *1 is

```
SEG1     CSEG
```

*2 is

```
SEG1     CSEG     AT 800H
```

with the relocation attribute of statement *1 replaced by that of statement *2, the relocation attribute of (1) becomes the default INBLOCK attribute which is different from the absolute attribute of (2). Thus, they are not linked and statement *2 will result in errors upon assembly.

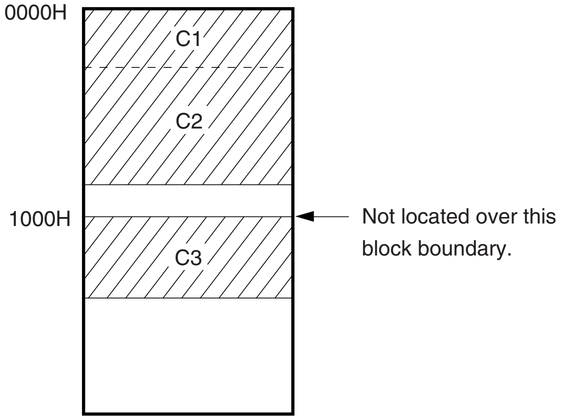
CSEG code segment CSEG

[Usage Examples]

Example 1. When INBLOCK or INBLOCKA attribute is specified

```
C1          CSEG        INBLOCK
            MOV         A, #5
            BR          L1
              .
              .
              .
L1:         SET1        PORT3.3
            RET
C2          CSEG        INBLOCK
            CLR1        MBE
            IN          XA, PORT4
            MOV         20H, XA
              .
              .
              .
C3          CSEG        INBLOCK
            MOV         A, @DL
            BRCB        !L2
              .
              .
              .
L2:         ADDS        A, #1
            END
```

The segments are located as follows upon linkage.



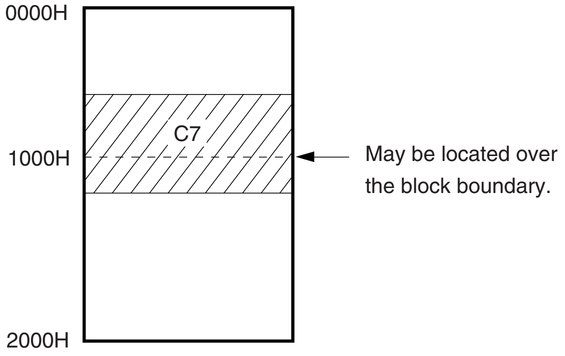
Segments ‘C1’, ‘C2’, ‘C3’ are located in the block[Note] upon linkage. They are not located over the block boundary.

Note INBLOCK : In the same block in 0000H to 3FFFH range
INBLOCKA : In the same block in the whole ROM space

Example 2. When XBLOCK or XBLOCKA attribute is specified

```
C7          CSEG        XBLOCK
S1:         MOV         XA, #50
            BR          LOOP
             .
             .
             .
LOOP:       MOV         A, #5
            BR          S1
             .
             .
             .
            END
```

If an address is not specified by the linker, the segments may be located as follows.

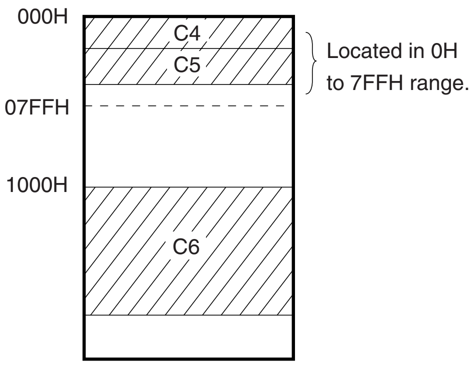


Segment ‘C7’ is located at any address[Note] upon linkage.

Note XBLOCK : Any address in 0000H to 3FFFH range
XBLOCKA : Any address in the whole ROM space

Example 3. When SENT attribute is specified

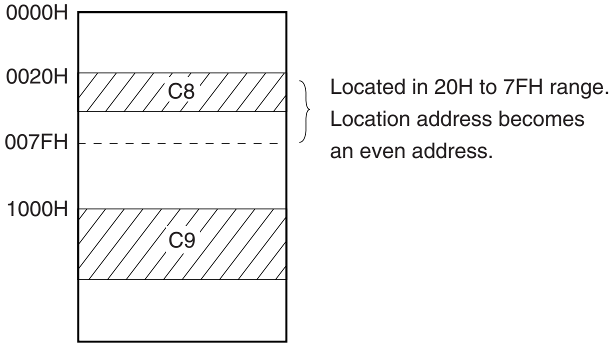
```
C4       CSEG      SENT
SUB1:    MOV       A, B
           .
           .
           .
         RET
C5       CSEG      SENT
SUB2:    SET1      PORT4.1
           .
           .
           .
         RET
C6       CSEG
         CALLF     !SUB1
           .
           .
           .
         CALLF     !SUB2
           .
           .
           .
         END
```



Segment ‘C4’ and ‘C5’ with SENT attribute specified are located in the 0H to 7FFH range.

Example 4. When IENT attribute is specified

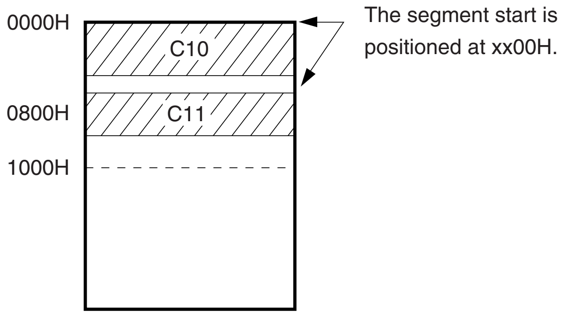
```
C8          CSEG        IENT
T1:         MOV         A, @DL
            INCS        L
CSUB2:      TCALL       SUB2
BSUB3:      TBR         SUB3
C9          CSEG
            GETI        CSUB2
            GETI        BSUB3
              .
              .
              .
            END
```



Segment ‘C8’ with IENT attribute specified is located in the 20H to 7FH range. The location address becomes an even address.

Example 5. When PAGE attribute is specified

```
C10       CSEG      SENT PAGE
          MOVT      XA, @PCDE
          BR        PCXA
           .
           .
           .
          RET
C11       CSEG      INBLOCK PAGE
          MOV       DE, #50
          BR        PCDE
           .
           .
           .
          END
```



The segment start is positioned at xx00H.

Segment ‘C10’ and ‘C11’ are located so that the start address becomes xx00H.

DSEG **data segment** **DSEG**

(2) DSEG (data segment)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
Segment name	DSEG	[Bank value] [AT absolute formula]	[;comment]

[Functions]

- DSEG pseudo-instruction instructs the assembler to start the data segment.
- Memory areas to be defined by DS pseudo-instruction before the segment definition pseudo-instruction (CSEG, DSEG) appears again following this pseudo-instruction will belong to the data segment. They will finally be reserved in the data memory.

[Applications]

- Describe DS pseudo-instruction mainly for the data segment defined by DSEG pseudo-instruction.

  The data segments are located in the data memory. Thus, instructions cannot be described in the data segments.
- In each data segment the data memory work area for use by the program is reserved using DS pseudo-instruction and a label is attached to the address of each work area.

  When the data memory work area is referred to in the source program, this label is used.

[Description]

- Segment names can be referred to as symbols.
- DSEG pseudo-instruction remains valid until the next segment definition pseudo-instruction or END pseudo-instruction appears.
- Segment names automatically become external definition symbols.
- The areas to be used can be switched by specifying the located data memory bank value. If this specification is omitted, bank 0 is used.
- In the case of μPD75000, the bank value is 0 to 15. In the case of all other unit types, the number of on-chip banks may differ. For details, refer to **APPENDIX A LIST OF ASSEMBLED RELEVANT UNIT DEVICE**.
- Specify the data segment start address by specifying ‘AT absolute formula’. If this specification is omitted, ‘AT 0H’ is set.

DSEG data segment DSEG

Example 1. DSEG0 DSEG AT 30H
Storage starts at address 30H of bank 0.

2. SEG1 DSEG 1 AT 20H
Storage starts at address 120H of bank 1.

Caution

Since 0F80H to 0FFFH of the data memory is reserved as the location protected area, specifying the data segment for this area using an absolute address will result in errors.

[Usage Example]

```
                  :
SEG0          DSEG          1 AT 30H
DATA1:DS            10
                  :
```

10-nibble area from address 130H of bank 1 is reserved.

ORG **origin** **ORG**

(3) ORG (origin)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
[Label:]	ORG	Formula	[; comment]

[Function]

- Sets the formula value specified by the operand to the location counter.

[Application]

- When locating the code or data segment at the particular address, specify ORG pseudo-instruction.

[Description]

- External reference name cannot be described in the formula.
- When using a symbol in the formula, define the symbol before ORG pseudo-instruction.
- When CSEG pseudo-instruction is defined using ‘AT absolute formula’ or CSEG pseudo-instruction is not used, ORG pseudo-instruction is used.

  Namely, ORG pseudo-instruction cannot be used for description in the relocatable segment.
- When using a list converter, ORG pseudo-instruction must be described using capitals in and after the 9th column of the source program.
- The location before change by the operand is assigned for the label in the symbol column on the line where ORG pseudo-instruction is described.

ORG **origin** **ORG**

Example 1.

```
ABS          CSEG         AT 10H
               :
             ORG          100H           ; (1)
               :
             ORG          $ + 40H        ; (2)
               :
             ORG          50H            ; (3)
               :
             ORG          XYZ            ; (4)
               :
XYZ          EQU          $
               :
```

(1) Location counter value = 100H

(2) Location counter value = current value + 40H

(3) Location counter value = 50H

(4) Error
Symbol ‘XYZ’ has not been defined before ORG pseudo-instruction.

2.

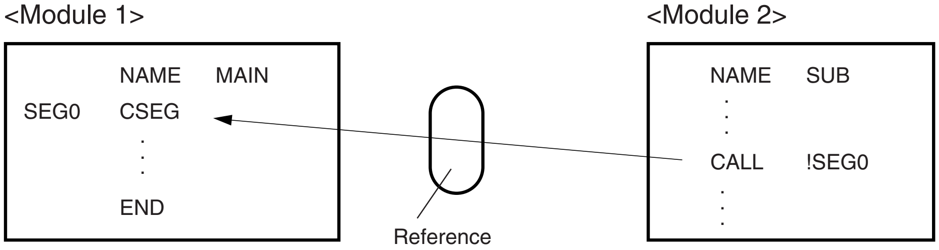
```
               :
REL          CSEG
               :
             ORG          10H            ; (1)
               :
```

(1) Error
ORG pseudo-instruction has been described in the relocatable segment.

4.3 PROGRAM LINKAGE PSEUDO-INSTRUCTIONS

Program linkage pseudo-instructions are used to clarify the relationships when the symbol defined by another module is referred to.

Let us look into the case where one program is generated separately in modules 1 and 2.

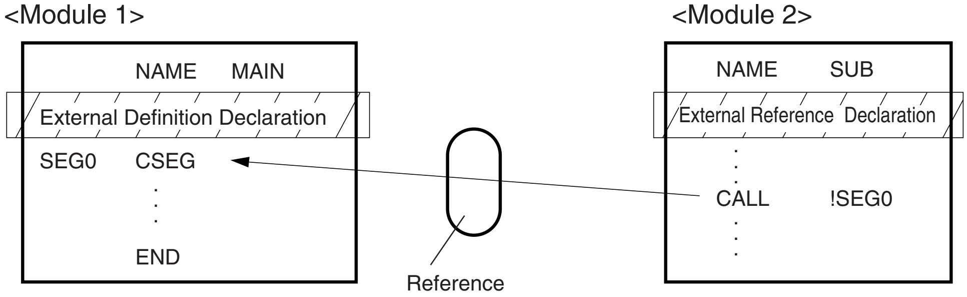


When referring in module 2 to the symbol defined in module 1, declarations are made in none of the modules and the symbol cannot be used. It is necessary to display “What to use” and “May use” in each module.

In module 1, the external definition (PUBLIC) declaration of the symbol that “the symbol may be referred to from another module” is made.

In module 2, the external reference (EXTRN) declaration of the symbol that “the symbol defined in another module is referred to” is made.

Only when two declarations of external reference and external definition are made effectively, the symbol can be referred to.

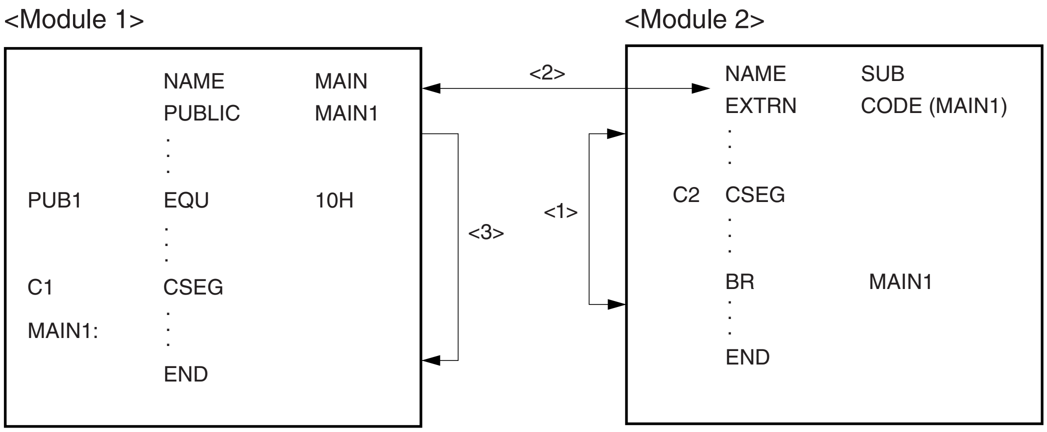


The following program linkage pseudo-instructions are used to set the above interrelationships.

- EXTRN pseudo-instruction to declare the external reference of symbol
- PUBLIC pseudo-instruction to declare the external definition of symbol

Symbol relations between modules are described referring to **Figure 4-3. Symbol Relations between Two Modules**.

Figure 4-3. Symbol Relations between Two Modules



```
            NAME        MAIN
            PUBLIC      MAIN1
             .
             .
             .
PUB1        EQU         10H
             .
             .
             .
C1          CSEG
             .
MAIN1:       .
             .
            END
```

```
        NAME        SUB
        EXTRN       CODE (MAIN1)
         .
         .
         .
C2      CSEG
         .
         .
         .
        BR           MAIN1
         .
         .
         .
        END
```

There are module 1 and module 2. They are named ‘MAIN’ and ‘SUB’, respectively.

- In module ‘SUB’ in Figure 4-3, external reference declaration is made in module ‘SUB’ using EXTRN pseudo-instruction because symbol ‘MAIN1’ defined in module ‘MAIN’ is referred to.
- In module ‘MAIN’, external definition declaration is made for symbol ‘MAIN1’ referred to from module ‘SUB’ is made using PUBLIC pseudo-instruction.

  The linker checks the interrelationships between the external reference and definition symbols.

NAME name NAME

(1) NAME (name)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
[Label:]	NAME	Module name	[; comment]

[Functions]

- The module name described for the operand is supplied to the object module output by the assembler.
- This pseudo-instruction can be omitted. If it is omitted, the primary source module file name becomes the module name.
- A characvter which cannot be used as the module name is replaced with ‘X’.

[Application]

- The module name is necessary for symbolic debugging using a debugger.

[Description]

- The module name cannot be described as an operand for other pseudo-instructions and instructions.
- ★ The module name is an alphanumeric string with thirty one or less characters (for details, refer to the [symbol description rules] in **3.3.3 Character Component Fields (1) Symbol column**).
- The name specified as the module name cannot be used as a symbol. The symbol with the same name as the module name cannot be defined.
- If more than one NAME pseudo-instruction is described in one module, an error results.

[Usage Example]

```
        NAME            SAMPLE          ; (1)
        DSEG
       :
       :
        CSEG
       :
       :
        END
```

(1) The module name is declared as SAMPLE.

Caution

If object file creation (-O option) has not been specified by assembler option, NAME pseudo-instruction specification has no meaning.

(2) PUBLIC (public)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
None	PUBLIC	Symbol [, ...]	[; comment]

[Function]

- Declares that the symbol described for the operand is the symbol referred to from another module.

[Application]

- If the symbol referred to from another module has been defined, be sure to declare external definition for the symbol using PUBLIC pseudo-instruction.

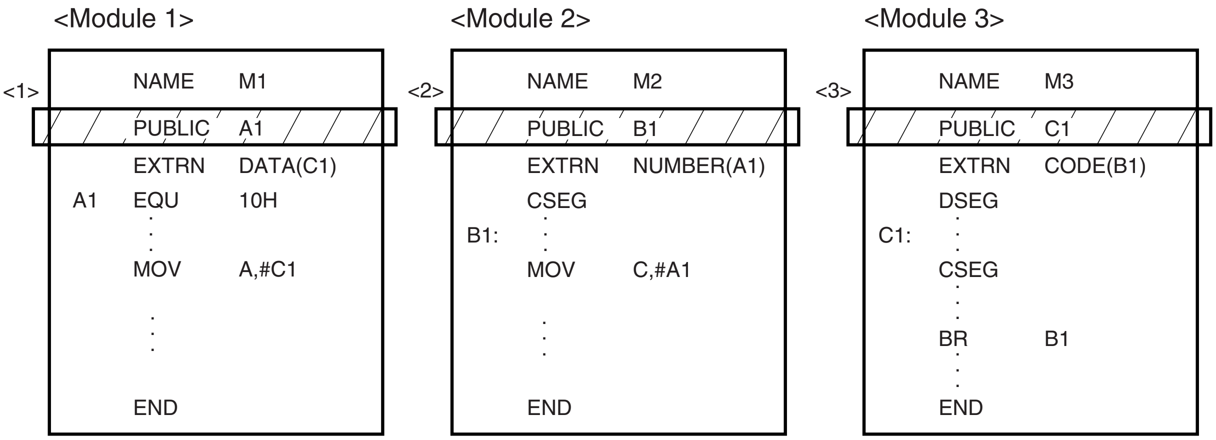
[Description]

- PUBLIC pseudo-instruction must be described before the symbol described is described in the operand column. Thus, it is recommended to describe PUBLIC pseudo-instruction in the module header.
- Two or more symbols divided by commas (,) can be specified for the operand.
- The symbol described for the operand must be defined in the same module.
- ★ The symbol is an alphanumeric string with thirty one or less characters headed by an alphabetic letter (for details, refer to the [symbol description rules] in **3.3.3 Character Component Fields (1) Symbol column**).

PUBLIC public PUBLIC

[Usage Example]

Example Example of program consisting of three modules



<Module 1>

```
<1>         NAME    M1
            PUBLIC  A1
            EXTRN   DATA(C1)
    A1      EQU     10H
             :
            MOV     A,#C1
             :
            END
```

<Module 2>

```
<2>         NAME    M2
            PUBLIC  B1
            EXTRN   NUMBER(A1)
            CSEG
    B1:      :
            MOV     C,#A1
             :
            END
```

<Module 3>

```
<3>         NAME    M3
            PUBLIC  C1
            EXTRN   CODE(B1)
            DSEG
    C1:      :
            CSEG
             :
            BR      B1
             :
            END
```

<1> Declares that symbol ‘A1’ is the symbol referred to from another module.

<2> Declares that symbol ‘B1’ is the symbol referred to from another module.

<3> Declares that symbol ‘C1’ is the symbol referred to from another module.

Caution

The segment name requires no PUBLIC declaration. The external reference name (the symbol declared by EXTRN pseudo-instruction), the name defined by SET pseudo-instruction and the specific address name code must not be described for the operand.

(3) EXTRN (external)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
None	EXTRN	Symbol CODE (Symbol [, ...]) DATA (Symbol [, ...]) BIT (Symbol [, ...]) PBIT (Symbol [, ...]) NUMBER (symbol [, ...])	[, ...] [; comment]

[Functions]

- Declares that the symbol described for the operand is referred to by this module.
- This symbol has been defined in another module.

[Applications]

- When the symbol defined in another module is referred to, be sure to declare external reference for the symbol using EXTRN pseudo-instruction.
- The external reference declared symbol can be described as a symbol without being defined in the module.

[Description]

- EXTRN pseudo-instruction must be described before the symbol referred to is described in the operand column. Thus, it is recommended to describe EXTRN pseudo-instruction in the module header.
- Two or more symbols divided by commas (,) can be specified for the operand.
- When the symbol attribute (CODE, DATA, BIT, PBIT, NUMBER) is specified, it becomes the symbol attribute of the symbol.
- The external reference name values are solved by the linker when linking. If no symbol attribute has been described, it cannot be checked upon assembly (for details, refer to **Table 3-2. Symbol Attribute Types**). When defining the external reference name using EXTRN pseudo-instruction, it is necessary to specify the symbol attribute.

  If symbol attribute is omitted, ‘NUMBER’ will be the symbol attribute.

  The symbol attribute specification procedure is shown in **Table 4-4. Symbol Attribute Specification Procedure**.

Table 4-4. Symbol Attribute Specification Procedure

Item No.	Attribute	Specification Procedure
1	CODE	• Label defined in the code segment • Name with the label having symbol attribute 'CODE' or '$' in the code segment defined using EQU pseudo-instruction
2	DATA	• Label defined in the data segment • Name with the label having symbol attribute 'DATA' or '$' in the data segment defined using EQU pseudo-instruction
3	BIT	• Name with the bit value defined using EQU pseudo-instruction
4	PBIT	• Name with the specific address name code defined using EQU pseudo-instruction with the bit operator, with a value in the range from FB0H.0 to FBFH.3 or from FF0H.0 to FFFH.3
5	NUMBER	• Name with the constant (except the bit value and specific address name code) defined using EQU pseudo-instruction

Example

<Module 1>

```
        NAME        M1
        PUBLIC      A1
C1      CSEG
A1:
        :
        :
        END
```

<Module 2>

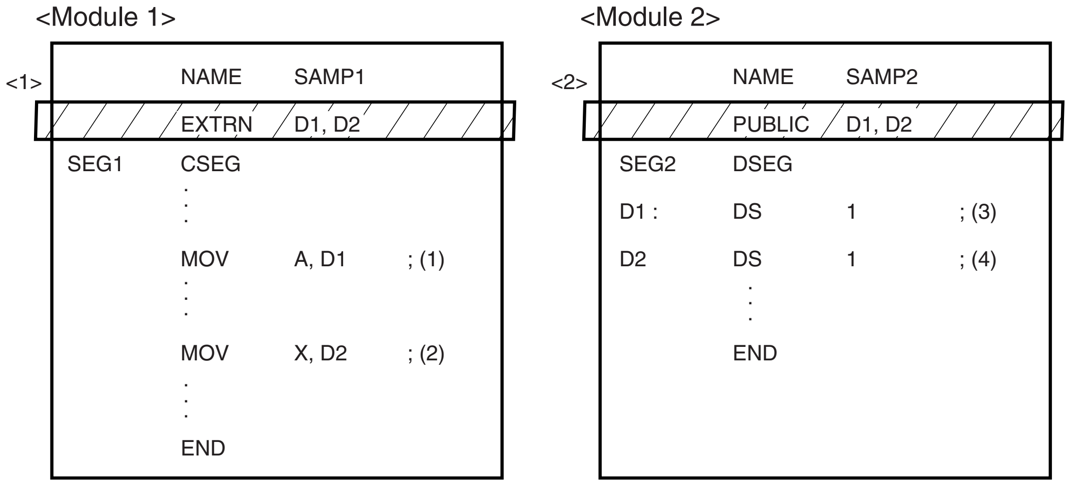
```
        NAME        M2
        EXTRN       CODE (A1)     ; (1)
C2      CSEG
        :
        :
        CALL        !A1
        :
        :
```

The attribute of symbol 'A1' in module 2 will become 'CODE' as a result of (1) EXTRN declaration.

EXTRN external EXTRN

[Usage Example]

Example



<Module 1>

```
<1>              NAME     SAMP1
                 EXTRN    D1, D2
       SEG1      CSEG
                 .
                 .
                 .
                 MOV      A, D1      ; (1)
                 .
                 .
                 .
                 MOV      X, D2      ; (2)
                 .
                 .
                 .
                 END
```

<Module 2>

```
<2>              NAME     SAMP2
                 PUBLIC   D1, D2
       SEG2      DSEG
       D1 :      DS       1          ; (3)
       D2        DS       1          ; (4)
                   .
                   .
                   .
                 END
```

<1> External reference is declared for symbols ‘D1’ and ‘D2’ referred to in (1) and (2).

<2> External definition is declared for symbols ‘D1’ and ‘D2’.

Both <1> and <2> enable more than one symbol to be described on one line.

(1) Symbol ‘D1’ is referred to.

(2) Symbol ‘D2’ is referred to.

(3), (4) Symbols ‘D1’ and ‘D2’ are defined.

Caution

Four operators ‘+’, ‘–’, ‘.’ , and ‘NOT’ can be used for the external reference name (for details, refer to **3.4.2 Operation Restrictions**).

4.4 SYMBOL DEFINITION PSEUDO-INSTRUCTIONS

Symbol definition pseudo-instructions are used to assign the name for the data which is to be used for source module description. This makes the data value meaning clear and the source module contents easy-to-understand.

These pseudo-instructions are used to notify the assembler of the name values for use in the source module.

These pseudo-instructions must be described before the reference symbol is described in the operand column. Thus, describe the name definition using the symbol definition pseudo-instruction in the module header.

EQU and SET pseudo-instructions are symbol definition pseudo-instructions.

EQU **equate** **EQU**

(1) EQU (equate)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
Name	EQU	Formula	[; comment]

[Function]

- Defines the value of formula specified by the operand and the attributes (symbol attribute and relocation attribute).

[Applications]

- The numeric data to be used in the source module is defined as the name and is described for the instruction operand in place of the numeric value.

  It is recommended to define as the name the numeric data which is frequently used in the source module.
- If the data value in the source module has been defined as the name, the data value can be changed by simply changing the operand value of the name.

[Description]

- When describing the name and label for the EQU pseudo-instruction operand, they must have been defined in the source module.
- If there is an error in the description in the symbol or mnemonic column of the statement with the name defined using EQU pseudo-instruction, the name is not registered.

  The statement which referred to the name will also be an error.
- In the case of a description error in the operand column of the statement with the name defined using EQU pseudo-instruction, name registration is carried out but 0 or an indeterminate value is assigned for the name value.
- The name defined using EQU pseudo-instruction cannot be redefined in the same source module.
- The symbol attribute of the name is the same as that of the operand.
- EQU pseudo-instruction can be described anywhere in the source program.
- When defining the symbol which becomes CODE attribute using EQU pseudo-instruction, the operand must be one that has been defined in the same segment.
- Only if the bit operator has been used for the specific address name code, the specific address name code can be described for the operand. If the code is described in all other cases, an error results.
- If definition of a new symbol is attempted using the symbol of CODE attribute defined by another segment for the segment, an error results.

EQU equate EQU

```
                  :
C1            CSEG
START :
                  :
C2            CSEG          AT 200H
EXAM          EQU           START
                  :
```

An error results.

An error occurs because symbol address calculations may not be carried out.

[Usage Example]

```
                  :
SUBR          EQU         7        ; (1)
SEG1          EQU         83H      ; (2)
                  :
              CALL        !SUBR
                  :
              BRCB        !SEG1
                  :
              END
```

Set 07H and 83H to (1) name ‘SUBR’ and (2) name ‘SEG1’, respectively. They will become NUMBER attribute.

(2) SET (set)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
Name	SET	Formula	[; comment]

[Functions]

- Defines the name having the formula value specified by the operand and the attributes (symbol and relocation attributes).
- The name can be described for the instruction code and pseudo-instruction operand.

[Applications]

- The variable used in the source module is defined as the name and is described for the instruction operand in place of numeric data (variable).
- When changing the name value in the source module, different numeric data can be defined for the same name using SET pseudo-instruction again.

[Description]

- When describing the name for SET pseudo-instruction operand, the name must have been defined in the source module.
- The external reference name and forward reference symbol cannot be described for SET pseudo-instruction operand.
- If there is an error in the description in the symbol or mnemonic column of the statement with the name defined using SET pseudo-instruction, the name is not registered. The statement which referred to the name will also be an error.
- In the case of a description error in the operand column of the statement with the name defined using SET pseudo-instruction, name registration is carried out but 0 is assigned for the name.
- PUBLIC declaration is disabled for the name defined using SET pseudo-instruction. The name is not output to the symbol table file generated by the object converter.
- The symbol attribute of the name is the same as that of the operand.
- SET pseudo-instruction can be described anywhere in the source program.
- When defining the symbol which becomes CODE attribute using SET pseudo-instruction, the operand must be one that has been defined in the same segment.
- Only if the bit operator has been used for the specific address name code, the specific address name code can be described for the operand. If the code is described in all other cases, an error results.

SET set **SET**

[Usage Example]

```
              :
IMMED    SET     5              ; (1)
         MOV     A, #IMMED      ; (2)
              :
IMMED    SET     10H-6          ; (3)
         MOV     A, #IMMED      ; (4)
              :
```

(1) Value 5 is supplied to name ‘IMMED’.
    This value remains valid just before description of (3).

(2) Value 5 of ‘IMMED’ has been transferred to the register.

(3) The value of name ‘IMMED’ is changed to 10H - 6 = 0AH.

(4) Value 0AH of ‘IMMED’ has been transferred to the register.

4.5 DATA DEFINITION AND AREA RESERVE PSEUDO-INSTRUCTIONS

Data definition pseudo-instructions are used to define constant data for use by the program.
The defined data value is generated as the object code.
Area reserve pseudo-instructions are used to reserve the memory area for use by the program.

(1) DB (define byte)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
[Label;]	DB	Formula Character string [, ...]	[; comment]

[Function]

- Initializes the memory byte-wise using the initial value specified by the operand.

[Application]

- When the constants (numeric and character constants) for use by the program is defined byte-wise, DB pseudo-instruction is described in the code segment.

[Description]

- The following two types of operands can be described as the initial value.

  <1> Formula
  The formula value is reserved as 8-bit data.
  If the formula value is greater than 8 bits, lower 8 bits are secured as data and an error is printed.

  <2> Character string
  A 7-bit ASCII code is reserved for one character and ASCII codes proportional to the number of characters are assigned sequentially for the memory.

- Up to a maximum of 16 operands divided using commas (,) can be described. If the operand is a character string, up to a maximum of 80 characters can be described for one operand.

  DB pseudo-instruction can be described only in the code segment (CSEG).

DB **define byte** **DB**

[Usage Example]

```
                NAME   SAMP1
CSEG1           CSEG
DATA1 :         DB     0A0H              ; (1)
DATA2 :         DB     0AFH-20H          ; (2)
WORD1 :         DB     'ABCD'            ; (3)
WORD2 :         DB     3*2, 'X', 'V'     ; (4)
DATA3 :         DB     132H              ; (5)
                 :
                 :
                END
```

(1) 1-byte area is initialized by 0A0H.

(2) 1-byte area is initialized by 0AFH to 20H, that is, 8FH.

(3) 4-byte area is initialized by character string 'ABCD'.
The area is assigned for the memory using ASCII codes 41H, 42H, 43H and 44H.

(4) 3-byte area is initialized by 3*2, 'X' and 'V'. The area is assigned for the memory using 06H, 58H and 56H.

(5) Because the number of bytes is greater than 1 byte, an error results.

(2) DS (define storage)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
[Label;]	DS	Absolute formula	[; comment]

[Function]

- Reserves the memory area proportional to the number of nibbles specified by the operand.

[Applications]

- DS pseudo-instruction is used to reserve the memory (RAM) area mainly for use by the program.
- If there is a label, the start address value of the reserved memory area is assigned for the label. In the source module, memory operation description is done using the label.

[Description]

- The contents of the reserved area are indeterminate.
- When describing a name and a label for the operand, they must be described for the absolute term previously defined in the source module.
- When a label is described in the symbol column, the label has the start address value of the reserved area.
- When the operand value is 0, no area is reserved.

DS   define storage   DS

[Usage Example]

```
           NAME     SUB1
DSEG0      DSEG     1 AT 10H
WORK :     DS       2               ; (1)
CSEG0      CSEG
           MOV      A,#5
           MOV      WORK.A
           MOV      WORK+1,A
           MOV      WORK1,A         ; (2)
             :
             :
WORK1 :    DS       2               ; (3)
           END
```

(1) 2-nibble work area is reserved. The reserved areacontents are indeterminate. Label ‘WORK’ is assigned for the start address.

(2) Because the operand symbol ‘WORK1’ has already been defined for (3) following this instruction, an error results.

(3) STKLN (stack length)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
None	STKLN	Absolute formula	[; comment]

[Function]

The area proportional to the nibble specified by the operand from the start address of the stack area indicated by reserved word 'STACK' is reserved as the stack area (thus, this is a pseudo-instruction which becomes valid only when SP is set using the reserved word 'STACK'). This pseudo-instruction reserves the stack area only in the module where it is described.

[Application]

- STKLN pseudo-instruction is used to reserve the stack area for use by the program.

[Description]

- The reserved area contents are indeterminate.
- STKLN pseudo-instruction can be described anywhere in the source program.
- Only the predefined labels or names can be described for the operands.
- STKLN pseudo-instruction becomes valid only when a value is set to the stack pointer using the reserved word STACK in the source program.

  Use the following procedure to set the value to the stack pointer.

  In the 75X Series/75XL Series, the addresses 0 to 0FFH in the data memory are the stack area, and there are some devices that also have the stack area at addresses except 0 to 0FFH. However, the stack pointer can only be set at addresses 0 to 0FFH for assembler package. The 8-bit register which holds the start address information of the STACK area is mapped as a stack pointer ataddress 0F80H of the data memory. It has thespecific address name code 'SP'. Since the SPcontents are indeterminate by the RES signal generation, it must be initialized to the specified value at the beginning of the program. The following two methods are available to set the value to the SP:

  1) Method of specifying using the absolute address
  2) Method of specifying using the reserved word 'STACK'

Each method has the following advantages and disadvantages.

STKLN stack length STKLN

1. Method of specifying using the absolute address

When specifying the stack pointer value using the absolute address, describe the source program as follows:

Example Stack pointer setting specifying the absolute address

```
;INITIALIZE  SP
TEST         CSEG
             MOV        XA, #00H
             MOV        SP,XA
```

As the stack pointer value is specified using the absolute address and assembled, the value can no longer be changed. To change the stack pointer value, reassembly and relinkage operations are necessary after the source program is revised.

For these reasons, if it is difficult to determine the stack pointer value, the following method should be used.

★ However, if the device has a stack area at addresses other than 0 to 0FFH in the data memory, specify the stack pointer value with an absolute address.

2. Method of specifying using the reserved word ‘STACK’

Stack pointer setting using the reserved word ‘STACK’ is carried out in two stages with the assembler and the linker.

To specify the stack pointer value with the reserved word ‘STACK’, describe the source program as follows:

Example Stack pointer setting using the reserved word ‘STACK’

```
;INITIALIZE  SP
TEST         CSEG
             MOV        XA, #STACK
             MOV        SP,XA
```

In this case, the reserved word ‘STACK’ value is determined upon linkage. In the stage of assembly, 00H is supplied as the default value to ‘STACK’.

In other words, because the initial value of the stack pointer can be set freely in the linker stage if a value has been set to the SP using the reserved word ‘STACK’, assembly is not required for changing the stack pointer. Thus, if the stack pointer value cannot be determined after the program development has been just started, programming using the reserved word ‘STACK’ is more efficient than SP specification using the absolute address.

Further, the reserved word STACK is closely related to the STKLN pseudo-instruction and linker stack option used to reserve the stack area.

Addresses 0 to 0FFH of the data memory can be used not only as the stack area but also as the normal data area. In this case, the data memory used as the data area should not overlap the data memory used as the stack area. For that purpose, the linker can prevent the data and stack areas from overlapping using the assembler information concerning to what extent the data memory space is used as the stack area. This function is carried out by the STKLN pseudo-instruction of the assembler and the -SZ option of the linker.

These pseudo-instruction and link option judge the reserved word STACK value to be the SP start address. Thus, when setting the value to the SP, it is necessary to describe using the reserved word STACK.

STKLN stack length STKLN

[Usage Example]

Example STKLN pseudo-instructions of more than one module

<Module 1>

```
TEST1        CSEG
             MOV        XA, #STACK
             MOV        SP,XA
               :
             STKLN      12                 ← 12-Nibble Stack Area
               :                             Reserved in This Module
             END
```

<Module 2>

```
TEST2        CSEG                          ← No Stack Area
               :                             Reserved in This Module
DATA1        DSEG
               :
             END
```

<Module 3>

```
TEST3        CSEG
               :
DATA2        DSEG
             STKLN      20                 ← 20-Nibble Stack Area
               :                             Reserved in This Module
             END
```

One program is divided into three modules (modules 1 to 3). In module 1, the value is set to the stack pointer using the reserved word ‘STACK’ and the depth of the STACK area used in the module 1 is declared by the STKLN pseudo-instruction.

In module 2 where the stack area is not used, the STKLN pseudo-instruction is not used.

In module 3, the depth of the STACK area used in the module is declared by the STKLN pseudo-instruction.

When these three modules are linked by the linker, the most significant address of the stack area is indicated by ‘STACK’ and 12 nibbles declared in module 1 and 20 nibbles declared in module 3 add up to 32 nibbles for the stack area. However, the value of 32 nibbles may be greater than that required for the stack area by the program. Thus, the -SZ option of the linker has been devised to adjust the size of the stack area for the entire program. For details of the -SZ option, refer to **5.4.4 Description of Linker Option** of the Manual for **Operation**.

STKLN **stack length** **STKLN**

Caution

When reserving or referring to the stack area using the STKLN pseudo-instruction and the reserved word STACK, the stack area is set to memory bank 0 by the linker.

In the case of unit types having the stack area set to other than bank 0, set the memory bank value to be used for the reserved word SBS and set the stack pointer (the offset value in the 8-bit bank) to the reserved word SP. In this case, reserve the stack area using the DSEG pseudo-instruction. (The stack area cannot be reserved over more than one memory bank.)

When the stack area is reserved using the DSEG pseudo-instruction, the stack pointer and the stack size cannot be reset by the -SK and -SZ options upon linkage.

Example Setting the stack pointer to memory bank 2

```
 ;INITIALIZE SBS, SP
TEST          CSEG
              MOV           A, #2
              MOV           SBS,A
              MOV           XA,#00H
              MOV           SP,XA
;
;STACK        AREA
DI            DSEG          2 AT 0
              DS            100H
```

When setting the stack area to memory bank 0, the STKLN pseudo-instruction and the reserved word STACK are effective irrespective of the unit types.

4.6 BRANCH INSTRUCTION AUTO SELECT PSEUDO-INSTRUCTIONS

Four 75X Series/75XL Series unconditional branch instructions are used to directly describe the branch destination address as the operand. They are “BR $addr”, “BRCB !caddr”, “BR !addr” and “BRA !addr1”.

<1> “BR !addr1” is a 3-byte instruction which can be branched to all addresses.

<2> “BR !addr” is a 3-byte instruction which can be branched in the range of 0 to 3FFFH.

<3> “BRCB !caddr” is a 2-byte instruction which can be branched to the inside of the same block (4 Kbytes of X000H to XFFFH) as where the BRCB instruction exists.

<4> “BR $addr” is a 1-byte instruction which can be branched in the range of (current program counter – 15) to (current program counter + 16).

Therefore, to generate a program having a high memory efficiency, it is necessary to select the instruction according to the branch destination range although considering the branch destination range when describing the branch instruction is extremely troublesome.

Under these circumstances, pseudo-instructions have been devised to select the branch instruction enabling the assembler to automatically select the branch instruction having a minimum number of bytes according to the branch destination range. They are branch instruction auto select pseudo-instructions.

BR branch BR

(1) BR (branch)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
[Label :]	BR	Formula	[; comment]

[Function]

- The assembler automatically selects the branch instruction having a minimum number of bytes according to the formula value range specified by the operand and generates the corresponding object code.

This function is called optimization.

[Applications]

- It is troublesome to take into account the branch destination range when describing the branch instruction. Thus, if it is difficult to select the describable branch instruction, use the BR pseudo-instruction.
- When the describable branch instruction is clearly identifiable, describe the corresponding instruction. This helps to decrease the assembly time as compared to when describing the BR pseudo-instruction.

[Description]

- This pseudo-instruction can only be used in the code segment.
- Only ‘CODE’ and ‘NUMBER’ symbol attributes can be described in the formula.
- The assembler can optimize the BR pseudo-instruction only when a symbol having CODE attribute (including the label defined using colon ‘:’ in the source program code segment) is described for the operand.
- When the NUMBER attribute symbol (the symbol with the absolute value assigned using EQU pseudo-instruction) or the absolute value is described for the operand, this instruction is replaced with a 3-byte instruction[Note].
- In the case of unit types having no 3-byte instructions, the BR pseudo-instruction is replaced with 2-byte instruction “BRCB !caddr”.

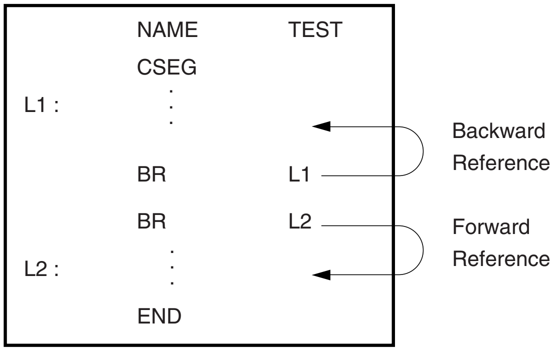
Refer to the name and label using the methods listed in **Table 4-5. Name and Label Reference Methods**.

Note When the ROM size is 16 KB or less ... “BR !addr”
When the ROM size is more than 16 KB ... “BRA !addr1”

Table 4-5. Name and Label Reference Methods

Item	Reference No.	Method Description
1	Backward	The name and label referred to as reference the operand have been defined in the precedingsource module.
2	Forward	The name and label referred to as reference the operand have been defined in the subsequent source module.

<Source Module>



```
            NAME        TEST
            CSEG
L1 :          :
            BR          L1
            BR          L2
L2 :          :
            END
```

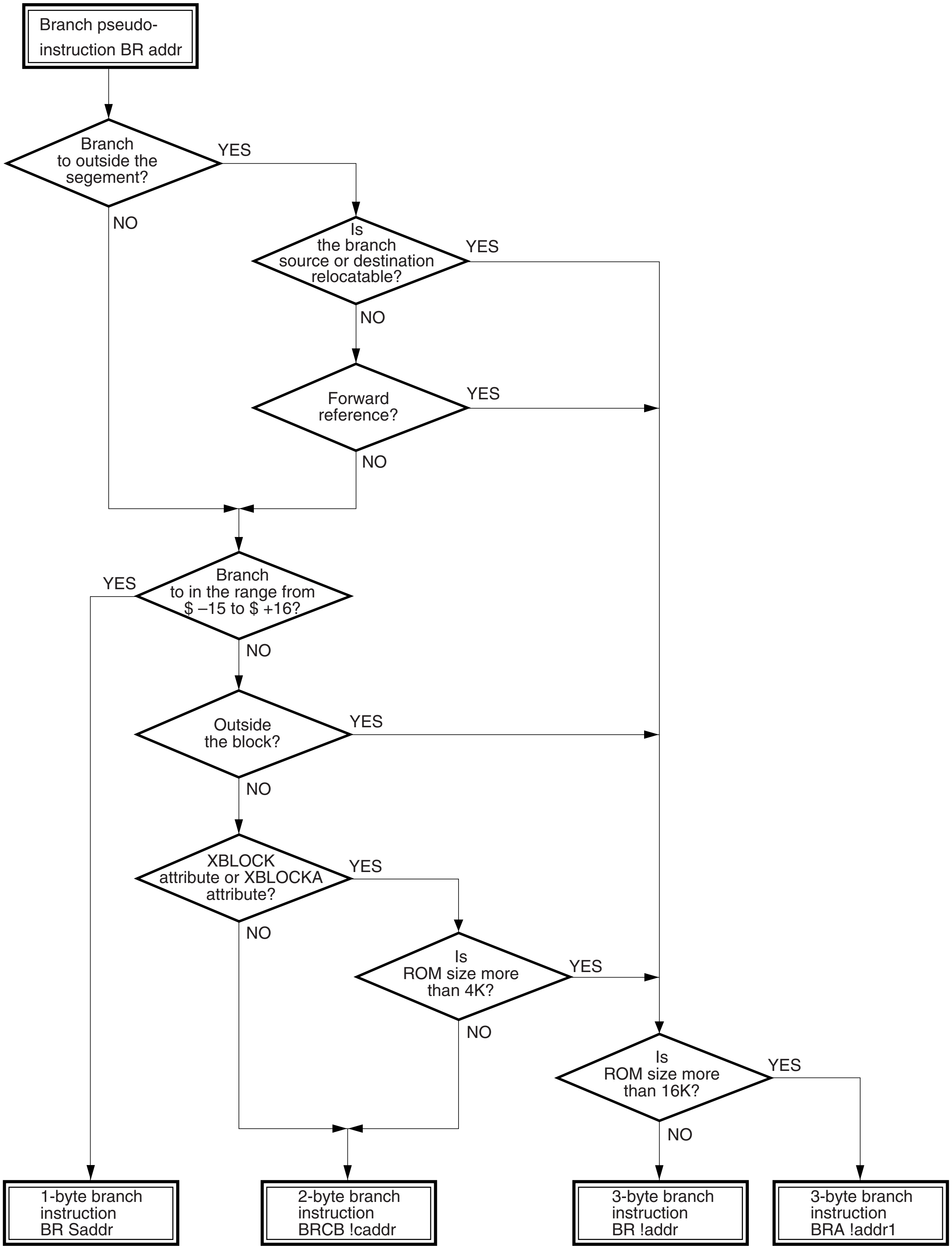
Summary of optimization procedure is shown below.

Table 4-6. Optimization Procedure

<table>
<tr><th colspan="2" rowspan="3">Branch
Destination</th><th colspan="3">Inside Segment</th><th colspan="3">Outside Segment</th></tr>
<tr><th colspan="2">XBLOCK attribute XBLOCKA attribute</th><th rowspan="2">INBLOCK attribute INBLOCKA attribute IENT attribute SENT attribute Absolute</th><th colspan="2">Both branch source and destination are absolute segments</th><th rowspan="2">Either branch source or destination is a relocatable segment</th></tr>
<tr><th>ROM size is 4 KB or more</th><th>ROM size is 4 KB or less</th><th>Backward reference</th><th>Forward reference</th></tr>
<tr><td colspan="2">No branch</td><td colspan="3">BRCB !caddr</td><td colspan="3"></td></tr>
<tr><td colspan="2">In the range from $ –15 to $ +16</td><td colspan="4">BR $addr</td><td colspan="2"></td></tr>
<tr><td rowspan="2">Outside the range from $ –15 to $ +16</td><td>Inside the block</td><td></td><td colspan="3"><b>BRCB !caddr</b></td><td colspan="2"></td></tr>
<tr><td>Outside the block</td><td colspan="6">BRA !addr1 (ROM size is more than 16K)
BR !addr (ROM size is 16K or less)</td></tr>
</table>

BR branch BR

The assembler executes optimization as follows:


Branch pseudo-instruction BR addr
Branch to outside the segement?
YES
NO
Is the branch source or destination relocatable?
YES
NO
Forward reference?
YES
NO
Branch to in the range from $ –15 to $ +16?
YES
NO
Outside the block?
YES
NO
XBLOCK attribute or XBLOCKA attribute?
YES
NO
Is ROM size more than 4K?
YES
NO
Is ROM size more than 16K?
YES
NO
1-byte branch instruction BR Saddr
2-byte branch instruction BRCB !caddr
3-byte branch instruction BR !addr
3-byte branch instruction BRA !addr1

BR branch BR

Caution

- BR pseudo-instruction jumps to itself (BR $, LBL: BR LBL, etc.), the assembler generates the code of 2-byte BRCB instruction (1-byte relative branch instruction cannot branch to itself).

  If the BR pseudo-instruction is described on the block boundary (XFFEH, XFFFH), the generated BRCB instruction can refer to itself beyond the block boundary. Thus, an error results in this case.
- If the BR pseudo-instruction is located on the block boundary (XFFFH), (PHASE ERROR) may occur on the subsequent label definition lines.

Do not describe the BR pseudo-instruction on the block boundary.

Example When BR pseudo-instruction is located at a block boundary

```
ADRS      OBJECT      SOURCE      STATEMENT

0100                  C1          CSEG   AT     100H
  :                                 :
  :                                 :
0FFE            5020                     BR     FORWARD

1000      60          LBL:        NOP

          * * *     ERROR      #100 PHASE ERROR      ← 3-byte branch changed to 2-byte
  :                                 :                  branch because the instruction is
  :                                 :                  located on the block boundary (LBL
101F      60         FORWARD : NOP                     should have been located at
                                                       address 1001)
```

★ • If available branch instructions vary among some devices of the same subseries of the 75X Series (due to ROM size differences, for example) do not use the BR directive.

Example	µPD75116H	(ROM 16 Kbytes)	· · ·	"BR !addr" ("BRA !addr1" is unavailable.)
	µPD75P117H	(ROM 24 Kbytes)	· · ·	"BRA !addr1"

Assume that the µPD75P117H is the target device. In this condition, if the program is assembled after a BR directive is described, the code for "BRA !addr1" will be created. Then, if the ROM size of the µPD75P117H has been set to 16 Kbytes with the stack bank select register (SBS), the program does not run correctly with actual applications although it runs with "BRA !addr1" in an in-circuit emulator seemingly without problem (With "BR !addr", it runs correctly).

BR branch BR

[Usage Example]

```
C0        CSEG      INBLOCK   ;
LABEL :   NOP                 ;  LABEL is the CODE attribute symbol.
NUM       EQU       100H      ;  NUM is the NUMBER attribute symbol.
          BR        LABEL     ;  Optimization is executed.
                                    (CODE attribute symbol)
          BR        NUM       ;  No optimization is executed.
                                    (NUMBER attribute symbol)
          BR        100H      ;  No optimization is executed.
                                    (Operand has absolute value.)
```

4.7 VECTOR ENTRY TABLE DEFINITION PSEUDO-INSTRUCTIONS

When carrying out interrupt servicing, the interrupt start address is set to the vector table corresponding to each vectored interrupt.

Programming can be carried out efficiently by specifying the memory bank, register bank or the symbol indicating the interrupt service start address to the vector table.

This also applies to high-speed interrupt service.

Caution

The interrupt start address to be set to the vector table has 14 bits. Thus, segments located at and after 4000H cannot be set as the interrupt start address.

VENTn **vector entry table** **VENTn**

(1) VENTn (vector entry table)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
None	VENTn	MBE = $\left\{\begin{matrix}0\\1\end{matrix}\right\}$, RBE = $\left\{\begin{matrix}0\\1\end{matrix}\right\}$, Start address	[; comment]

n: Number 0 to 7

n = 0 .... Address
n = 1 .... Address 2
n = 2 .... Address 4

n = 7 .... Address 14

Depending on the assembled unit type, 0 to 7 numbers may not be used or the RBE may not be incorporated. For details, refer to **APPENDIX A LIST OF ASSEMBLED RELEVANT UNIT DEVICE**.

[Function]

- The 75X Series/75XL Series has the vector table to set the interrupt start address corresponding to each vectored interrupt at addresses 0H to 0FH of the program memory.

  In this area the area with a total of 16 bits of the memory bank enable flag (MBE), register bank enable flag (RBE) and the entry address is automatically reserved.

  This function is done by the VENTn pseudo-instruction. For details of the entry address area, refer to **Figure 4-4. VENTn Pseudo-Instruction and Program Memory**.

[Application]

- This instruction is defined when the vector entry table is used.

[Description]

- Since the VENTn pseudo-instruction is used to define the information inevitable to start the program, it must be described before all mnemonics, segment definition pseudo-instruction and area reserve pseudo-instruction.
- Start address information in particular upon internal reset at addresses 0H and 1H is necessary for any program. These two addresses are defined by the VENT0 pseudo-instruction. In addition, the VENTn pseudo-instruction can be used as a normal program memory.
- Usable vectored interrupts differ partly depending on the assembled unit type.

  Thus, describable VENTn pseudo-instructions also differ depending on the assembled unit type. For details, refer to **APPENDIX A LIST OF ASSEMBLED RELEVANT UNIT DEVICE**.

- Memory banks and register banks are not incorporated depending on the assembled unit type. In the case of those types, be sure to set 0 to MBE and RBE.
- When a list converter is used, the VENTn pseudo-instruction must be described in capitals from the 9th column of the source program (the VENTn code is not changed by the list converter).
- The interrupt start address must be in the range from 0H to 3FFFH.

Figure 4-4. VENTn Pseudo-Instruction and Program Memory

Address	7	6	0	
0000H	MBE	RBE	Internal Reset Start Address (upper 6 bits)	... VENT0
			Internal Reset Start Address (lower 8 bits)	
0002H	MBE	REB	INTBT/INT4 Start Address (upper 6 bits)	... VENT1
			INTBT/INT4 Start Address (lower 8 bits)	
0004H	MBE	RBE	INT0 Start Address (upper 6 bits)	... VENT2
			INT0 Start Address (lower 8 bits)	
0006H	MBE	RBE	INT1 Start Address (upper 6 bits)	... VENT3
			INT1 Start Address (lower 8 bits)	
0008H	MBE	RBE	INTSIO Start Address (upper 6 bits)	... VENT4
			INTSIO Start Address (lower 8 bits)	
000AH	MBE	RBE	INTT0 Start Address (upper 6 bits)	... VENT5
			INTT0 Start Address (lower 8 bits)	
000CH	MBE	RBE	INTTPG Start Address (upper 6 bits)	... VENT6
			INTTPG Stat Address (lower 8 bits)	
000EH	MBE	RBE	INTKS Start Address (upper 6 bits)	... VENT7
			INTKS Start Address (lower 8 bits)	

[Usage Example]

```
          VENT0     MBE=0,    RBE=0,    START  ........ Located at 0H.
          VENT2     MBE=1,    RBE=1,    SUBI   ........ Located at 4H.
            :
            :
CI        CSEG
START :   BR        SUB3
          BR        SUB4
            :
            :
          END
```

4.8 GETI INSTRUCTION TABLE DEFINITION PSEUDO-INSTRUCTIONS

The GETI instruction can convert the following four types of instructions into 1-byte instructions.

- Subroutine call instruction in the range of 0 to 3FFFH
- Branch instruction to the range of 0 to 3FFFH
- Any 2-byte, 2-machine cycle instruction (except BRCB and CALLF instructions)
- Combination of two 1-byte instructions

The number of bytes can be decreased and programs with high memory efficiency can be generated by converting frequently used instructions using the GETI instruction.

TCALL table call TCALL

(1) TCALL (table call)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
[Label:]	TCALL	Call address	[; comment]

[Function]

- The necessary data is reserved to execute the call instruction by the GETI instruction.

[Application]

- When it is desired to execute 2 or 3-byte call instructions “CALLF !faddr” and “CALL !addr” to within 16 KB with 1 byte using the GETI instruction, the TCALL pseudo-instruction is described in the GETI instruction table (20H to 7FH).

[Description]

- TCALL pseudo-instruction is used to reserve GETI instruction data corresponding to the call instruction.
- In the source program, the GETI instruction is described in place of a 2 or 3-byte call instruction and the defined address of the corresponding TCALL pseudo-instruction is described in the operand column.
- The call address must be in the range of 0 to 3FFFH.

Cautions

1. TCALL pseudo-instruction is a code segment and can only be described with the following relocation attributes.

<1> IENT attribute

<2> AT attribute with the location counter at an even address in the range of 20H to 7FH

2. Do not describe “CALL !addr” and “CALLF !faddr” in the GETI instruction table (20H to 7FH).

3. Call instruction “CALLA !addr1” to within 64 KB cannot be executed by the GETI instruction.

[Usage Example]

```
C2   CSEG   IENT
S1:  TCALL  SUB1  ← Actual Call Address
S2:  TCALL  SUB2
C22  CSEG
     GETI   S1
     GETI   S2
```

(2) TBR (table branch)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
[Label:]	TBR	Branch address	[; comment]

[Function]

- The necessary data is reserved to execute the branch instruction by the GETI instruction.

[Application]

- When it is desired to execute 2 or 3-byte branch instructions “BRCB !caddr” and “BR !addr” to within 16 KB with 1 byte using the GETI instruction, the TBR pseudo-instruction is described in the GETI instruction table (20H to 7FH).

[Description]

- TBR pseudo-instruction is used to reserve GETI instruction data corresponding to the branch instruction.
- In the source program, the GETI instruction is described in place of a 2 or 3-byte branch instruction and the defined address of the corresponding TBR pseudo-instruction is described in the operand column.
- The branch address must be in the range of 0 to 3FFFH.

Cautions

1. TBR pseudo-instruction is a code segment and can only be described within the following relocation attributes.

   <1> IENT attribute

   <2> AT attribute with the location counter at an even address in the range of 20H to 7FH
2. Do not describe “BR !addr” and “BRCB !caddr” in the GETI instruction table (20H to 7FH).
3. Branch instruction “BRA !addr1” to within 64 KB cannot be executed by the GETI instruction.

[Usage Example]

```
C3        CSEG   IENT
BSUB3:    TBR    SUB3  ← Actual Branch Address
C33       CSEG
          GETI   BSUB3
```

4.9 ASSEMBLY END PSEUDO-INSTRUCTION

The assembly end pseudo-instruction instructs the assembler to end the source module. This pseudo-instruction is always described at the end of the source module.

END   end   END

(1) END (end)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
None	END	None	[; comment]

[Function]

- The end of the source module is declared to the assembler.

[Application]

- The END pseudo-instruction is described at the end of the source module.

[Description]

- The assembler assembles the source modules until the END pseudo-instruction appears.

[Usage Example]

```
        NAME  SUB
        DSEG
          :
        CSEG
          :
        END           ; (1)
```

(1) The END pseudo-instruction is described at the end of the source module.

[MEMO]

★ CHAPTER 5 MACRO

In this chapter, how to use the macro function will be described. This function is useful if a source program needs to include the same series of instructions repeatedly.

5.1 OUTLINE OF MACROS

The macro function is useful if the same series of instructions must be included in a source program repeatedly. The concept of this function is that a part of the program, which users define as a macro body with the MACRO and ENDM instructions, is copied to every location where the macro body is to be referenced.

A macro, different from a subroutine, is to be used to improve source program coding efficiency. The following section summarizes the characteristics of subroutines and macros to clarify their appropriate usage.

(1) Subroutine

- A subroutine holds a series of instructions which will be executed many times when the program runs. The assembler encodes it to machine code only once for each subroutine.
- To reference a subroutine, only a subroutine call instruction must be described (In many cases, instructions that set arguments are also required before and after the reference). Therefore, effective use of subroutines can lead to a high program memory utilization.
- Subroutines also contribute to the creation of structured programs if programmers define every processing having a specific function as a subroutine whenever possible. (A structured program permits easy grasp of the configuration of the program and facilitates program design.)

(2) Macro

- The basic concept of the macro function is to replace an instruction with a series of referenced instructions. The instructions to be replaced must be enclosed with the MACRO and ENDM instructions. The instructions so enclosed will be copied (developed and encoded) to the location of each referencing instruction.
- The assembler, when it detects a macro reference, develops the referenced macro body, while replacing the tentative parameter in the macro body with the actual parameters at the time of referencing, and converts the macro body into machine language.
- A macro can include parameters.
  For example, assume that there are some instruction groups which do the same operation and differ only in the data to be specified as operands. In this case, tentative parameters for the data should be assigned in a macro definition section. Then, if macro names and actual parameters are prepared at the time of macro referencing, plural instruction groups which differ from one another in a limited section only can be integrated into a single macro.

In short, subroutines should be used to save the memory size and/or structure of a program, while macros should be used to improve the coding efficiency.

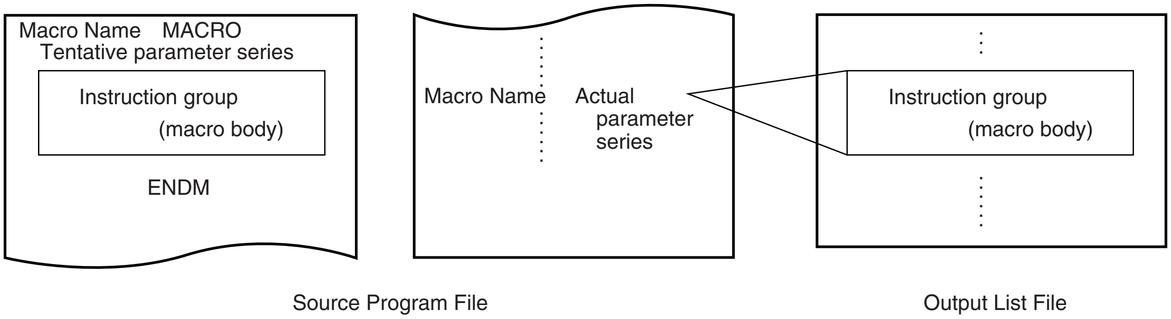
5.2 MACRO TYPES

There are two macro types. This section outlines each type, and the details will be discussed in sections **5.4** through **5.9**.

(1) Macro (MACRO)

A macro is referenced with its macro name which must have been predefined for the corresponding instruction group. Parameters can also be provided at this time.

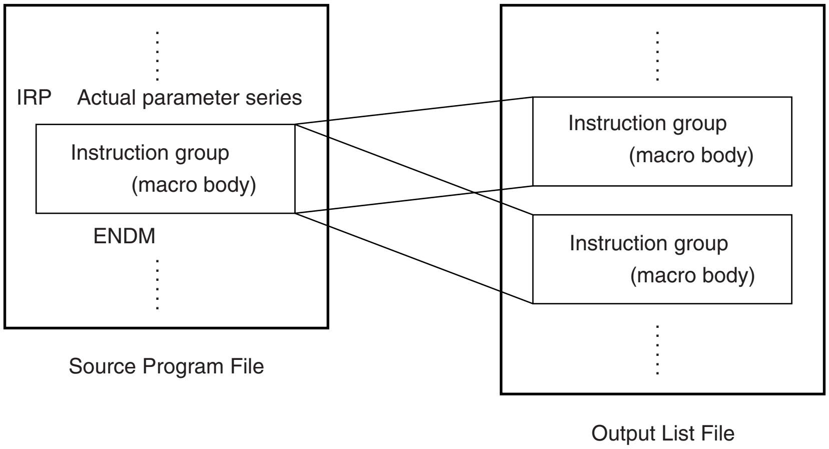
Figure 5-1. Concept of Macro



(2) Repeat macro (REPT, IRP, IRPC)

An instruction group which is declared as a macro will be developed at plural locations of the output file repeatedly. A macro of this type is suited for instruction groups whose contents are almost the same except for very small parts and that must be executed one after the other. In such case, specify the common part as a macro and the differences as parameters. The macro will be developed to the specified locations while changing parameter values at each development process.

Figure 5-2. Concept of Repeat Macro



The number of macro development processes is equal to the number of actual parameters.

5.3 MACRO RULES

5.3.1 Macro Definition Rules

(1) Macro body

A macro body is the section enclosed with a MACRO, REPT, IRP, or IRPC instruction and an ENDM instruction.

A macro body may include any kind of text except a macro definition and LODM instruction. If the LODM instruction is included in a macro body, the macro cannot be recognized as a macro.

When a MACRO instruction is detected, the lines after the MACRO line until the line preceding ENDM will be treated as a macro body.

(2) Maximum number of macros and local symbols

Macro bodies and local symbols are stored in the memory area. The maximum number of macros and local symbols that can be included in a source program, therefore, differs depending on the usable memory space.

(3) Redefinition of macro

If a macro name is defined which coincides with a macro name previously defined for a different macro, the macro contents for the new macro name overwrite the previously defined macro contents.

(4) Nesting

Nesting is the state in which a jump section is inserted in another jump section. The following four types of nesting are allowed within a macro.

Nesting to another macro: A macro body includes a macro referencing instruction.
Nesting of repeated macros: A repeated macro is described in the macro body of another repeated macro.
Nesting of include files: A $INCLUDE instruction is described in an include file.
Nesting of $IF blocks: A $IF instruction is described in the $IF block.

The maximum nesting level is 32, including the $IF, $SWITCH, and $INCLUDE instructions. However, nesting of $INCLUDE statements may be limited by the maximum number of files that can be opened on the operating system used.

If the nesting level becomes higher than level 32, the assembler aborts the development for higher-level nesting macros and proceeds with the line that follows, displaying “nest overflow”.

The maximum macro reflexive call level is also 32. If a nest overflow occurs, the assembler aborts the reflexive call macro and proceeds with the next line.

Caution Depending on the macro body definition size, the maximum nestable level may be less than 32.

(5) Tentative parameters

For the macros defined with the MACRO instruction or the repeated macros, parameters can be given when the macro is developed.

To do so, a tentative parameter must be described, when the macro is defined, to the location in the macro body where the actual parameter needs to be replaced. The actual parameter must be specified for the tentative parameter as an operand of the macro referencing instruction.

The tentative parameter SET definition is not allowed.

A tentative parameter can be combined with a character string with an ampersand “&” as shown below.

BR LOOP & PRM1 → BR LOOP2	Tentative parameter : PRM1
	Actual parameter : 2

This “&” will be ignored when the macro is developed. If “&” needs to be used for other purposes, describe “&&” to prevent misinterpretation.

Reserved words, predefined macro names, and predefined SET symbols must not be used as tentative parameter names. If such names are used, the macro will not be recognized as a macro and an error message will be displayed.

Predefined tentative parameter names written in comment lines will not be recognized as tentative parameters.

(6) Tentative parameter list

A tentative parameter used in a macro body must be declared in the MACRO instruction operand column. This operand description is called tentative parameter list. One or more symbols can be written in a tentative parameter list if they fit within the same line.

Each symbol must be separated from other symbols with a comma. If there are errors in the description of tentative parameters, the corresponding macros will not be registered.

5.3.2 Macro Reference Rules

A macro which is defined with the MACRO instruction is referenced by its macro name; a repeated macro is referenced when the definition is completed, that is, when an ENDM is executed.

(1) Actual parameter

Items (a) through (e) below can be described as actual parameters that are used when a macro is referenced. Note that use of reserved words as actual parameters is prohibited.

In the actual parameter list, parameters must be separated with commas (,). If there is a space character before or after a comma (,), the space character is not regarded as part of the actual parameter name. A space character that neither proceeds nor follows a comma is treated as part of the actual parameter name.

(a) Numeric constants
Binary, octal, decimal, hexadecimal constants can be used.

(b) Predetermined symbols
These symbols are those which have been defined with the SET pseudo instruction and the D option. For the SET pseudo instruction, refer to **4.4 (2) "SET"**. For the D option, refer to the **Operation** manual.

(c) Tentative parameters
If a macro referencing instruction is included in a macro definition section, actual parameters can be passed between macros by describing the same tentative parameter in the macro referencing operand column.

(d) Character string
If a character string needs to be sent as an actual parameter, write the string as is or enclose it in quotation marks (').
The maximum number of characters in a string is 128 excluding the quotation marks at both ends. A single quotation mark must not be included as part of an actual parameter. Therefore, if a quotation mark needs to be used as part of a parameter name, write two quotation marks in a row. If a character string includes characters whose ASCII codes are lower than 20H, an error message will be displayed and macro referencing will not be executed.

(e) Formula
Formulae can also be used for actual-parameter specification, in which case, those which meet the formula condition are retrieved and used as actual parameters.

(2) Macro development

A macro will be developed at the location (line) which references the macro.

5.4 OUTLINE OF MACRO INSTRUCTIONS

Various macros can be defined in the source program.
The following macro-related instructions are provided.

Table 5-1. List of Macro Instructions

Macro Instruction Type	Macro Instruction
Macro definition instruction	MACRO/EXITM/ENDM
External macro declare instruction	LODM
Macro instruction	Macro name
Repeat macro instruction	REPT, IRP, IRPC
Global symbol declare instruction	GLOBAL

5.5 MACRO DEFINITION INSTRUCTIONS

A macro definition instruction specifies an instruction group to be defined as a macro, and names it (defines a macro name).

MACRO **macro** **MACRO**

(1) MACRO (macro)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
Name	MACRO	[Tentative parameter list]	[; comment]
			[; comment]
	<Macro body>		
			[; comment]
[Label:]	ENDM		[; comment]

Two or more tentative parameters can be specified by separating them with commas (,). These parameters must fit within one line.

[Function]

- The statements, which are between the MACRO statement and the ENDM statement, are registered as a macro, and what is specified in Name is assigned as the name of the macro. Once it is registered, the name works as an instruction. To reference the registered macro, describe the macro name.

[Application]

- Define a frequently used series of statements in the source program as a macro. Then, describe the macro name predefined for the macro (reference a macro) so that the corresponding macro body is developed.

[Description]

- The definition of a macro can be made anywhere in the source program unless it is later than a macro referencing instruction or it is in the macro body section.
- A comment described in the MACRO statement line will not be registered as a macro body.
- If an error is included in a macro name—for example, no macro name is specified or the macro name specified coincides with a reserved word—the macro corresponding to such macro name will not be registered.
- If a symbol is included in an ENDM statement, the section before that symbol is registered as the macro body. Although describing a character string in the ENDM operand column causes an error, the corresponding macro will be registered correctly.

EXITM **exit from macro** **EXITM**

(2) EXITM (exit from macro)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
<MACRO instruction, Repeat macro (REPT, IRP, IRPC) instruction statements>			
	<Macro body>		
	EXITM		[; comment]
	<Macro body>		
[Label:]	ENDM		[; comment]

[Function]

Macro development is aborted immediately when an EXITM instruction is detected during macro development.

[Application]

- This function is mainly used when the conditioned assemble (Refer to **6.4 Conditioned Assemble Control Instruction**) function is used in the macro body defined with the MACRO instruction.
- If plural conditioned assemble functions are used in combination in a macro body, exit from the macro forcibly, or sections that should not be assembled may also be assembled. The EXITM instruction should be used in such case.

[Description]

This instruction can be described only in a macro body. Describing this instruction anywhere except a macro definition part causes an error.

If this instruction is detected during macro referencing, the macro development underway will be aborted and the statement that follows the ENDM instruction will be processed next.

If the macro development underway is a nested one, only the macro development at the current level will be aborted and the macro on the next lower level will be developed next.

If an EXITM instruction is detected during the macro body development for a repeat macro instruction, the macro body development process will be aborted immediately, and the statement that follows the ENDM instruction will be processed next. If the macro development is a nested one, processing moves on to the process on the next lower level.

If a character other than a tab or space character precedes the word EXITM on an EXITM line, the EXITM processing will be continued although an error occurs.

If a character string is described in the operand column, the EXITM processing will be continued although an error occurs.

(3) ENDM (end macro)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
<MACRO instruction, Repeat macro (REPT, IRP, IRPC) instruction statements>			
			[; comment]
	<Macro body>		
			[; comment]
[Label:]	ENDM		[; comment]

[Function]

- This instruction declares the end of a macro definition.

[Application]

- The ENDM instruction must always be described at the end of a series of macro statements that follow a MACRO, REPT, IRP, or IRPC instruction.

[Description]

- This instruction indicates completion of a macro body that began with a MACRO, REPT, IRP, or IRPC instruction.
- In the case of a repeated macro, a macro development process starts immediately when an ENDM instruction is detected.
- If a symbol is included in an ENDM statement, the section that precedes the symbol will be registered as a macro body. Although describing a character string in the ENDM operand column causes an error, that macro will be registered as it.

5.6 EXTERNAL MACRO DECLARE INSTRUCTION

The macros defined within the source program file are called internal macros. Apart from this, the macros which are stored in separate files and are referenced from the source program file are called external macros. By preparing general-purpose macros as external macros in separate files, these macros can be utilized from various source programs.

To define a series of operations as an external macro, use the MACRO instruction. With one file, only one macro can be defined. If there are two or more macros in a file, only the definition for the first macro is valid.

LODM **load macro** **LODM**

(1) LODM (load macro)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
	LODM	External macro name [, ······]	[; comment]

Two or more external macro names can be specified on the same line by separating them with commas (,). These names must fit within one line.

[Function]

- This instruction enables the macro stored in a file to be referenced from the source program.

[Application]

- Use this instruction when a macro stored in a separate file needs to be referenced from the source program.

[Description]

- The LODM instruction declares external macros. To develop external macros, the macro must be referenced by its name in the same way as the macros defined with the MACRO instruction. Because the mechanism is the same as the macro definition with the MACRO instruction, an LODM instruction cannot be described in a macro definition part. If it is described there, an error will occur and the LODM instruction will be invalidated.
- The file name for an external macro must be in the form “ExternalFileName.m”. Therefore, a reserved name for macros cannot be used for the primary name of the file. In addition, the name of the MACRO instruction for external macros must match the primary name.
- If there is an error in external macros, only illegal macros will be invalidated.

Remark When an external macro file is retrieved, the following directories are checked in that order.

(1) The directory that includes the source file
(2) The directory specified by the I option (The I option can specify up to eight paths.)
(3) The directory defined with the environment variable “MACLIB”

LODM load macro LODM

Caution

If MS-DOS™ or PC DOS™ is used, all file names must consist of eight characters or less. Therefore, even if the S option is specified, the first eight characters of a macro name will be the external macro name to be specified with the LODM instruction. For this reason, the macro referencing name and the macro definition name both must consist of eight or less characters and must match for the beginning eight characters. If the S option is not specified, the beginning eight characters are valid for both macro name definition and referencing.

If the NCA option is specified, uppercase and lowercase characters are treated as different characters for macro names, but not for LODM-specified external macro names. However, the macro referencing name must match the macro definition name.

As for the S/NS option and CA/NCA option, refer to the **Operation** manual.

Example **1.** Difference in definition between S option (extending valid symbol length from 8 to 31 characters) and NS option

Source program

```
        LODM      ABCDEFGH                  ··· (1)
;
        ABCDEFGH_I              10H,  20H   ··· (2)
;
        END
```

External macro content (ABCDEFGH.m) file

```
ABCDEFGH        MACRO    P1,  P2   ··· (3)
                MVI      A,  P1
                ADI      A,  P2
                ENDM
```

Remarks **1.** If either the S option is not specified or the NS option is specified, external macros are defined as and referenced with “ABCDEFGH”.

2. If the S option is specified, external macros are defined as “ABCDEFGH”. Therefore, the external macro referencing will be invalid. (See (2) in the illustration above.)

2. Difference in definition between NCA option (distinguishing uppercase and lowercase characters) and CA option

Source program

```
        LODM      ABCDEFGH                  ··· (1)
;
        abcdefgh  10H, 20H                  ··· (2)
;
        END
```

External macro content (ABCDEFGH.m) file

```
ABCDEFGH        MACRO    P1,  P2   ··· (3)
                MVI      A,  P1
                ADI      A,  P2
                ENDM
```

Remarks **1.** If either the CA option is specified or the CA, NCA options are not specified, external macros are defined and referenced.

2. If the NCA option is specified, external macros are defined as “ABCDEFGH” with uppercase characters. Therefore, the external macro referencing will be invalid. (See (2) in the illustration above.)

5.7 MACRO INSTRUCTION

This instruction calls predefined or declared macros and develops their contents.

macro instruction

(1) Macro Instruction

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
[Label [:]]	Macro name	[Actual parameter list]	[; comment]

One or more actual parameters can be specified on the same line by separating them with commas (,). These parameters must fit within one line.

[Function]

- This instruction references a predefined macro and develops its macro body while replacing the tentative parameters (predefined when the macro is defined) with the actual parameters.

[Application]

- A macro instruction is used when referencing a macro body is required.

[Description]

- Describe a predefined reference macro name in the mnemonic column and actual parameters in the operand column. A statement must be one line. Place commas (,) between parameters if there need to be two or more parameters. The macro name must have been defined prior to the statement which includes a macro reference in the source program, in either internal or external macro.
- If a character string is described in the symbol column, it is treated as the label for the macro name in the mnemonic column. The macro name in the mnemonic column is used to reference a macro.
- The colon (:) at the end of a label can be omitted.
- If the number of actual parameters is smaller than the number of tentative parameters, null strings (zero-length character string) are set for the tentative parameters that remained unfilled. Null strings are set also when some actual parameters are omitted.
- If on the opposite, the number of actual parameters is larger than the number of tentative parameters, actual parameters that do not have replacements are ignored. This does not cause an error.
- Incorrect description of the referenced macro name causes an error (Refer to the **Operation** manual).
- A space character before or after an actual parameter is ignored.

5.8 REPEAT MACRO INSTRUCTIONS

An instruction of this type repeatedly develops an instruction pattern upon declaration.
There are three instructions for this type as shown below.

- REPT instruction
- IRP instruction
- IRPC instruction

REPEAT repeat REPEAT

(1) REPT (repeat)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
	REPT	Formula	[; comment]
	<Macro body>		
[Label:]	ENDM		[; comment]

[Function]

- This instruction repeatedly develops a macro body by the number of times specified with the operand value. The macro body will be developed at the location where the REPT instruction is defined.

[Application]

- This instruction, together with the ENDM instruction, is used in the source program to repeatedly describe a series of statements in a row.

[Description]

- Describe a formula which determines the number of development times (less than or equal to 1023), in the operand column. If the formula includes a symbol, set the value for the symbol with the SET instruction.
- If there is an undefined symbol in the formula, an error message will be displayed and the repeat macro instruction will be skipped.
- If the formula value is over 1023, the assembler develops the macro body 1023 times and moves on to the next processing while issuing an error message.
- If repetition of more than 1023 times is needed, nest the REPT processes.
- If the formula value is 0, the macro body will not be developed at all.
- Note that a macro body must not include any macro definition, LODM instruction, or INCLUDE instruction.
- In the macro body for a repeat macro instruction, other repeat macro instructions such as IRP and IRPC can be defined if necessary. Also, reference statements for the macro can be included there.
- The maximum nesting level of repeated macros, but not limited to repeated macros, is 32, including nesting of macro statement, INCLUDE statement, IF statement, and CASE statement. If the nesting level is over 32, the assembler does not develop macros of over level 32 and issues an error message.
- In the list file, the value of the operand formula is displayed as a hexadecimal 4-digit number (XXXX) in the STNO column.

(2) IRP (indefinite repeat)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
	IRP	Tentative parameter, [Actual parameter list]	[; comment]
	<Macro body>		
[Label:]	ENDM		[; comment]

One or more actual parameters can be specified on the same line by separating them with commas (,). These parameters must fit within one line.

[Function]

- This instruction develops the macro body by the number of times equal to the number of actual parameters. The tentative parameter in the macro body will be replaced with each actual parameter which is taken out one by one from top to bottom in the actual parameter list specified here.

[Application]

- This instruction, together with the ENDM instruction, is used when it is required to describe in the source program a series of statements only a limited numeral section of which is different.

[Description]

- If actual parameters are omitted, the assembler replaces the tentative parameter with a null string and develops the macro body only once.
- If an error is included in the IRP operand description, the macro body will not be developed.
- In the macro body for a repeat macro instruction, other repeat macro instructions such as REPT and IRPC can be defined if necessary. Also, reference statements for the macro can be included there.
- The maximum nesting level of repeat macros, but not limited to repeat macros, is 32, including nesting of macro statement, INCLUDE statement, IF statement, and CASE statement. If the nesting level is over 32, the assembler does not develop macros over level 32 and issues an error message.

(3) IRPC (indefinite repeat of character)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
	IRPC	Tentative parameter, [Character string]	[; comment]
	<Macro body>		
[Label:]	ENDM		[; comment]

Describe in the operand column a tentative parameter, a comma (,), and a character string. This all must fit in one line. Note that the character string must not be enclosed with quotation marks. For this reason, do not include in the character string a space character or semicolon (:).

[Function]

- This instruction, when a macro is developed, replaces the tentative parameter in the macro body with one character in the character string specified in the operand column. The macro body will be developed by the number of characters consisting of the character string, and the character for replacement shifts one by one for each development.

[Application]

- This instruction, together with the ENDM instruction, is used when it is required to describe in the source program a series of instructions only one character of which is different.

[Description]

- If actual parameters are omitted, the assembler replaces the tentative parameter with a null string (0-length character string) and develops the macro body only once.
- If an error is included in the IRPC operand description, the macro body will not be developed.
- In the macro body for a repeat macro instruction, other repeat macro instructions such as REPT and IRPC can be defined if necessary. Also, reference statements for the macro can be included there.
- The maximum nesting level of repeated macros, but not limited to repeated macros, is 32, including nesting of macro statement, INCLUDE statement, IF statement, and CASE statement. If the nesting level is over 32, the assembler does not develop macros over level 32 and issues an error message.

5.9 GLOBAL SYMBOL DECLARE INSTRUCTION

This instruction enables a symbol declared in the macro, which is called local symbol, to be referenced in other processing as well as macro development.

GLOBAL global GLOBAL

(1) GLOBAL (global)

[Description Format]

Symbol column	Mnemonic column	Operand column	Comment column
	GLOBAL	Symbol name [, ······]	[; comment]

Two or more symbol names can be specified on the same line by separating them with commas (,). They must fit within one line.

[Function]

Usually, a symbol defined in a macro is valid only once for the development of the macro body and is called local symbol. However, there may be cases where the symbol needs to be used in other operations. A symbol that can also be referenced is called global symbol. To change a local symbol into a global symbol, declare so with the GLOBAL instruction. The symbol name specified in this instruction's operand column will become a global symbol after this instruction is executed.

For the valid range of symbols, refer to **3.3.3 Character Configuration Fields (1) Symbol Column [Valid range of symbol in macro]**.

[Application]

By executing this instruction, a symbol which was defined in a macro and valid only at macro development can be referenced in other processing.

[Description]

Unless otherwise declared, a macro treats a symbol as a local symbol. Therefore, to use it as a global symbol, declaration is required with the GLOBAL instruction. There are two typical such cases.

- A macro needs to reference a SET symbol located outside the macro or to change the value.
- A macro includes both a jump instruction and its destination. The macro is referenced only once. The label of the destination should not be modified.

The GLOBAL instruction can be described either within or outside a macro. If it is described in a macro, the symbol will be globalized when the macro is referenced.

A global symbol must not coincide with any macro names previously defined. In addition, tentative parameters in macro definition must not be declared as a global symbol.

If there is a character other than a tab or space before the word GLOBAL, a syntax error will be caused. Even so, however, the GLOBAL instruction will be normally executed.

CHAPTER 6 CONTROL INSTRUCTIONS

In this chapter, the types and functions of control instructions which describe in the source program will be described.

6.1 GENERAL DESCRIPTION OF CONTROL INSTRUCTIONS

Control instructions are used to instruct the assembler precisely on operations and are described in the source program.

They are not used for object code creation.

The following control instructions are available.

Table 6-1. Control Instruction Table

Control Instruction Type	Control Instruction
Include control instruction	INCLUDE
Assembly list control instruction	TITLE, LIST, NOLIST, EJECT
★ Conditional assemble control instruction	IFDEF/ELSE/ENDIF, IF/ELSE/ENDIF, SWITCH/CASE/BREAK/DEFAULT/ENDS

Like pseudo-instructions, the control instructions are described in the source program.

It is necessary to describe '$' (dollar mark) in the 1st column.

```
$ [ △ ] control instruction
↑
1st column
```

6.2 INCLUDE CONTROL INSTRUCTION

The include control instruction is used to cite another source module file in the source module.

The load required for source program description can be alleviated by using the include control instruction efficiently.

INCLUDE **include** **INCLUDE**

(1) INCLUDE (include)

[Description Format]

```
$ [ △ ] INCLUDE = File name
$ [ △ ] IC = File name          ;  Abbreviated form
```

1st column

[Function]

- The specified file contents are inserted and expanded on the specified line onward.

[Application]

- A series of relatively large statements to be described commonly in two or more source modules is arranged into one file (include file).

  When it becomes necessary to cite the series of statements in each source module, the required include file name is specified by the INCLUDE control instruction.

  This alleviates source module description work.

[Description]

- Dollar mark ($) is described in the 1st column.
  Only one blank space or TAB code can be input to separate '$' from 'INCLUDE'.
- When the file name is specified, the path name (drive name and directory name) where the include file is stored can be specified. If the path name is omitted, the following paths are searched for in this order as the candidates of the paths where the include file has been stored.

  <1> Path where the source module file has been stored
  <2> Path which has been specified by the -I option (refer to the **Operation** Manual) upon assembler startup
  <3> Path which has been specified by environment variable "INC75X"

  Since the details depend on the OS, refer to the **Operation** Manual.

- Include file nesting is possible at only thirty two levels (the nesting means the specification of another include file in the include file).
- When the END pseudo-instruction is described in the include file, assembly is stopped. Thus, do not normally describe the END pseudo-instruction in the include file.

Example

<Source Program>

```
            NAME      SAMPLE
            EXTRN     L1, L2
            PUBLIC    L3
$           INCLUDE=EQU.INC :(1)
            CSEG
              :
            END
```

<Include File>

EQU.INC

```
SYMA    EQU     10H
SYMB    EQU     20H
          :
SYMZ    EQU     100H
```

(1) ‘EQU.INC’ has been specified as the include file. If this source program is assembled, the include file contents will be expanded as follows.

```
            NAME      SAMPLE
            EXTRN     L1, L2
            PUBLIC    L3
$           INCLUDE=EQU.INC :(1)

SYMA        EQU       10H
SYMB        EQU       20H
              :
SYMZ        EQU       100H

            CSEG
              :
            END
```

The contents of include file ‘EQU.INC’ have been expanded.

6.3 ASSEMBLY LIST CONTROL INSTRUCTIONS

Assembly list control instructions are used to provide instructions preventing title or list output with respect to the assembly list generated by the assembler.

They are NOLIST, LIST, TITLE and EJECT control instructions.

TITLE **title** **TITLE**

(1) TITLE (title)

[Description Format]

```
$ [ △ ] TITLE = ‘Character string’
$ [ △ ] TT = ‘Character string’      ; Abbreviated form
```

1st column

[Function]

- The TITLE pseudo-instruction specifies the character string to be printed in the title part of the assembly list header.

[Applications]

- This pseudo-instruction is specified to display in the assembly list the title which clearly indicates the assembly list contents.
- The assembly list contents can be at a glance by printing the title on each page.

[Description]

- Up to a maximum of 60 characters are valid in a character sting. The 61st and subsequent characters are omitted.
- When the TITLE control instruction is specified, the list undergoes line feed and the character string specified by the TITLE control instruction is printed on the page after line feed.
- If the TITLE control instruction is not specified, the assembly list title column will be left blank.

TITLE   title   TITLE

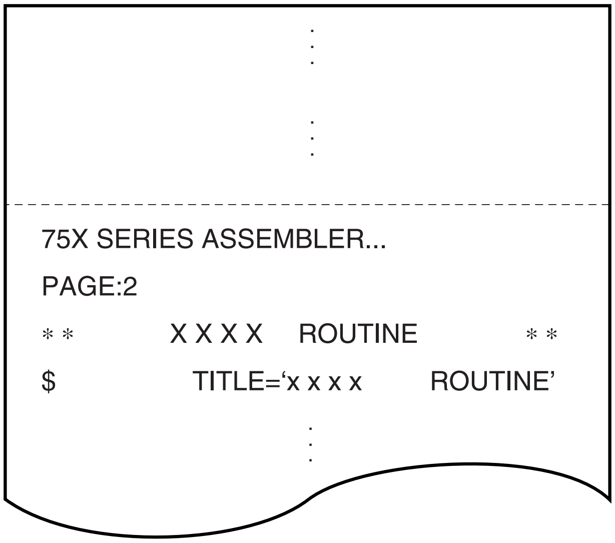
[Usage Example]

Example

<Source Module>

```
                        :
$            TITLE='x x x x     ROUTINE'
                        :
```

The assembly list is as follows:



```
                        :

                        :
- - - - - - - - - - - - - - - - - - - - - - - -
75X SERIES ASSEMBLER...
PAGE:2
* *          X X X X    ROUTINE           * *
$            TITLE='x x x x      ROUTINE'
                        :
```

<Page is turned.>

(2) NOLIST (no list)

[Description Format]

```
$ [ △ ] NOLIST
$ [ △ ] NOLI                    ; Abbreviated form
↑
```

1st column

[Functions]

- The NOLIST control instruction instructs the assembler on the assembly list output stop position.
- The statements generated between NOLIST control instruction and the next LIST control instruction are assembled but they are not output in the assembly list.

[Application]

- The NOLIST control instruction is used to limit the list output volume.

[Description]

- When the NOLIST control instruction is described, the dollar mark ($) is described in the 1st column.
  Only one blank space or TAB code can be input to separate '$' from 'NOLIST'.
- The NOLIST control instruction is intended to stop the assembly list output, not to stop the assembly operation.
- If the LIST control instruction is specified after the NOLIST control instruction, the statements generated after the specified LIST control instruction will be output in the assembly list again.
- If the NOLIST control instruction is omitted, the LIST control instruction is regarded as having been specified.
- When a list converter is used, describing the NOLIST control instruction will prevent the list from being converted correctly.

NOLIST **no list** **NOLIST**

[Usage Example]

```
        NAME    SAMP1
$       NOLIST              ; (a)
D1      EQU     10H
D2      EQU     11H
        :
D20     EQU     20H
$       LIST                ; (b)
        CSEG
        :
        END
```

No output in the assembly list.

(a) Since the NOLIST control instruction has been specified, statements generated up to (b) LIST pseudo-instruction will not be output in the assembly list. The NOLIST control instruction itself is output.

(b) Since the LIST control instruction has been specified, subsequent statements will be output in the assembly list again. The LIST control instruction itself is not output.

(3) LIST (list)

[Description Format]

```
$ [ △ ] LIST
$ [ △ ] LI                          ;  Abbreviated form
```

↑
1st column

[Function]

- The LIST control instruction instructs the assembler on the assembly list output start position.

[Application]

- The LIST control instruction is used to reset the assembly list output stop state specified by the NOLIST control instruction to the assembly list output state.

  The assembly list output volume and print contents can be controlled by using the NOLIST and LIST control instructions in pairs.

[Description]

- When the LIST control instruction is described, the dollar mark ($) is described in the 1st column. Only one blank space or TAB code can be input to separate '$' from 'LIST'.
- If the LIST control instruction has been specified after the NOLIST control instruction, the line with the specified LIST control instruction onward will be output in the assembly list. The described LIST control instruction itself is not output in the assembly list.

[Usage Example]

- Refer to the usage example of NOLIST control instruction.

EJECT **eject** **EJECT**

(4) EJECT (eject)

[Description Format]

```
$ [ △ ] EJECT
$ [ △ ] EJ                    ;  Abbreviated form
↑
1st column
```

[Function]

- The EJECT control instruction instructs the assembler to turn the assembly list page.

[Application]

- This instruction is described at a position where the page should be turned in the source module.

[Description]

- When the EJECT control instruction is described, the dollar mark ($) is described in the 1st column. Only one blank space or TAB code can be input to separate ‘$’ from ‘EJECT’.
- The image of the EJECT control instruction itself is printed on the previous page.

EJECT   eject   EJECT

[Usage Example]

<Source Module>

```
                 :
             MOV     A, #1H
             BR      $
$            EJECT            ;(a)
             CSEG
                 :
             END
```

(a) The page is turned by the EJECT control instruction and the assembly list becomes as follows.

```
                 :
             MOV     A, #1H
             BR      $
$            EJECT
-------------------------------------← Page is turned.
             CSEG
                 :
             END
```

★ 6.4 CONDITIONAL ASSEMBLE CONTROL INSTRUCTIONS

The conditional assemble control instructions are used to change the switch setting of the conditional assemble to select whether a series of statements in the source module are or are not to be assembled.

Conditional assemble control instructions include the IFDEF/ELSE/ENDIF control instruction, IF/ELSE/ENDIF control instruction, and SWITCH/CASE/BREAK/DEFAULT/ENDS control instruction.

Efficient use of these instructions enables to assemble only the necessary statements in the program selectively, without modifying almost all sections in the source module.

IFDEF　　　if defined　　　IFDEF

(1) IFDEF (if defined)

[Description Format]

```
$   [ △ ] IFDEF              Symbol
          <THEN clause>
[ $ [ △ ] ELSE
          <ELSE clause>]
$   [ △ ] ENDIF
```

[Functions]

- The clause to be assembled, THEN or ELSE clause, is determined according to the symbol definition status.
- If a symbol is defined, the instruction described in the THEN clause will be assembled.
- If no symbol is defined, the instructions described in the ELSE clause will be assembled. If no ELSE statement line is described, the ENDIF statement will be processed next.

[Application]

- This instruction allows to modify the source statements that need to be assembled, without greatly changing the source module.
- The debug-purpose statements in the source module, which are used only during the program development stage, can be included in or excluded from the target program to be encoded to the machine code, according to the switch setting of the conditional assemble.

[Description]

- Be sure to locate the IFDEF statement and ENDIF statement at the same level and pair them up. In addition, the IFDEF and ENDIF statements must not be intervened by any unpaired the SWITCH and END statements, a macro definition section, or a repeated macro.
- Describing an ENDIF statement and omitting an ELSE statement, or vice versa, causes an error.

IF **if** **IF**

(2) IF (if)

[Description Format]

```
$   [ △ ] IF                  Formula
          <THEN clause>
[ $ [ △ ] ELSE
          <ELSE clause>]
$   [ △ ] ENDIF
```

[Functions]

- The clause to be assembled, THEN or ELSE clause, is determined according to the formula value.
- If the value is true (= other than 0), the instructions described in the THEN clause will be assembled.
- If the value is false (= 0), the instructions described in the ELSE clause will be assembled. If no ELSE statement is described, the ENDIF statement line will be processed next.

[Application]

- This instruction allows to modify the source statements that need to be assembled, without greatly changing the source module.
- The debug-purpose statements in the source module, which are used only during the program development stage, can be included in or excluded from the target program to be encoded to the machine code, according to the switch setting of the conditional assemble.

[Description]

- The value for the formula described in the operand column must be determined prior to the IF statement.
- If errors are included in the formula in the IF statement operand column, the ENDIF statement line will be processed next.
- Be sure to locate the IF statement and ENDIF statement at the same level and pair them up. In addition, the IF and ENDIF statements must not be intervened by any unpaired the SWITCH and END statements, a macro definition section, or a repeated macro.
- Describing an ENDIF statement and omitting an ELSE statement, or vice versa, causes an error.

(3) SWITCH (switch)

[Description Format]

```
$   [ △ ] SWITCH             Formula
$   [ △ ] CASE          Numeric value:
    [<Instruction group>]
[ $ [ △ ] BREAK]
[ $ [ △ ] CASE          Numeric value:
    [<Instruction group>]
$   [ △ ] BREAK]
                    :
[ $ [ △ ] DEFAULT
    [<Instruction group>]
$   [ △ ] BREAK]
$   [ △ ] ENDS
```

[Functions]

- This instruction calculates the formula in the SWITCH statement and jumps to the CASE label that matches the calculation result. The statements preceding the label are skipped.
- A CASE label, DEFAULT label, or ENDCASE instruction that appears during the macro development process will not be developed.
- At the end of the instruction group for one CASE or DEFAULT label, one BREAK statement is described.

  If a BREAK statement appears during the macro development process, the statements preceding the ENDS statement line will be skipped.
- As a label for when no matching CASE label is found, the DEFAULT label can be specified.
- If no matching CASE label is found, the statements preceding the DEFAULT or ENDS statement line will be skipped.
- As the number that can be described in the CASE label, a binary, octal, decimal, or hexadecimal constant between 0H and 0FFFFH is allowed.

[Application]

- This instruction allows to modify the source statements that need to be assembled, without greatly changing the source module.
- The debug-purpose statements in the source module, which are used only during the program development stage, can be included in or excluded from the target program to be encoded to the machine code, according to the switch setting of the conditional assemble.

SWITCH switch SWITCH

[Description]

- The instruction described in the CASE label line will not be developed.
- The value for the formula described in the operand column must be determined prior to the SWITCH statement.
- If errors are included in the formula in the SWITCH statement operand column, the ENDS statement line will be processed next.
- Be sure to locate the SWITCH statement and ENDS statement at the same level and pair them up. In addition, the SWITCH and ENDS statements must not be intervened by any unpaired the IDEFF and ENDIF statements, unpaired the IF and ENDIF statements, a macro definition section, or a repeated macro.
- Describing an ENDS statement only and omitting necessary counterpart statements causes an error. This also applies to the BREAK statement, CASE label, and DEFAULT label.
- If a CASE label is described posterior to a DEFAULT label, the CASE label is omitted and the macro development starts with the DEFAULT label.
- A CASE label must not include negative numeric values or formula.
- If two or more CASE labels with the same name are described, the one described earliest is enabled.
- If two or more CASE labels exist and one of them is not paired with a BREAK statement, an error is not caused and the instructions preceding the instruction group that corresponds to the next CASE label will be developed.

Example

```
$  SWITCH       P1
$  CASE         0 :
     BR  !labe10
$  CASE         1 :
     BR  !labe11
$  BREAK
$  CASE         2 :
     BR  !labe12
$  BREAK
$  ENDS
```

The example shown above will be developed as follows:

(1) If P1 = 2,
    BR !labe12.

(2) If P1 = 1,
    BR !labe11.

(3) If P1 = 0,
    BR !labe10 and BR !labe11.

[MEMO]

CHAPTER 7 ASSEMBLER PACKAGE UTILIZATION

In this chapter, some methods of how to effectively use the assembler package will be introduced.

7.1 ASSEMBLER PACKAGE UTILIZATION

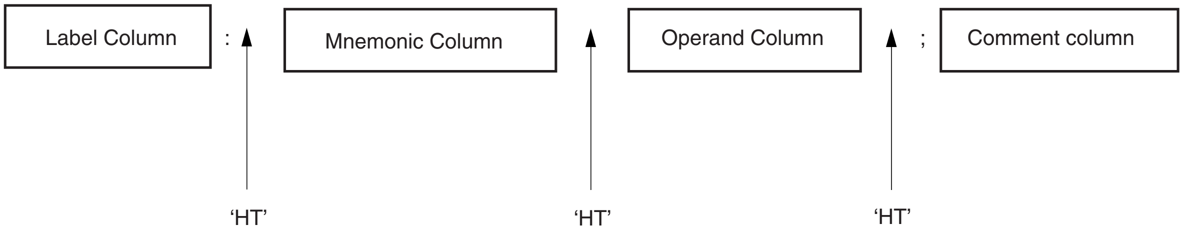
The product can be utilized in carrying out assembly operations using the assembler package.
Some of them are introduced below.

(1) Tabulation function

This function enables to facilitate source program generation and to make it easy to check the assembly list generated by assembly operation.

'HT' code is inserted before the mnemonic column, at the beginning of the operand column and before the semicolon (;) indicating the beginning of the comment column.

Insertion of the 'HT' code helps to make it easy to check each column of the source program and assembly list.



(2) Assembler option specification

It is quite troublesome to specify the option on each command line upon assembler program startup. It is easier to describe the necessary options in the parameter file.

Example **1.** If the map list file is not to be generated, describe the -NKM and -NP options in the link parameter file.
2. If the assembly list output is to be limited, use the NOLIST or LIST control instruction according to requirements.

(3) Description for data definition

Description for data definition is made at the start of the module header.

For example, let us suppose that although the value frequently used in the program has been assigned for the name by the EQU pseudo-instruction, it is now necessary to change the value for some reason.

In that case, all that must be done is to change the operand value of the name defined by the EQU pseudo-instruction. If no name has been defined by the EQU pseudo-instruction, changing the name must be carried out by looking for the name from the beginning to the end of the program.

(4) Comment description

If ‘what’ program has been generated by whom is described at the beginning of the source program, anyone can check the program easily.

In the case of a general-purpose module, the program can be made easier-to-understand if ‘what’ data is input, ‘what’ data is output and where the data is stored upon completion of module processing.

If there are small processing groups in the program, it may be more helpful to insert comment for each processing group.

A sample program is shown below.

```
$         TITLE = ‘A-D CONVERT’
$         XREF
;****************************************************
;***       A-D CONVERT PROGRAM                    ***
;***                          1988/XX/XX          ***
;***                          T.XXXX              ***
;****************************************************
```

Title and cross-reference list output are specified. Program contents and the data of program generation are described with comment.

```
           NAME     AD_MAIN
           EXTRN    CODE (SIOSUB, ADCONV)
           PUBLIC   TDATA, SEL15
           STKLN    10
           VENT0    MBE = 1, RBE = 1, MAIN
           VENT4    MBE = 1, RBE = 0, ADCONV

$         NOLIST
```

Assembly list from the next line to $LIST is not output.

```
;***      DATA AREA      ***
SEG0      DSEG     1 AT 10H
TDATA1:   DS       2
TDATA2:   DS       2
TDATA3:   DS       2
TDATA4:   DS       2
TDATA5:   DS       2
TDATA6:   DS       2
TDATA7:   DS       2

$         LIST
```

Assembly list from the next line is output.

```
;***      GETI TABLE      ***
SEG1      CSEG     IENT
SEL15:    SEL      MB15

;***      MAIN ROUTINE    ***
SEG2      CSEG     INBLOCK
MAIN;     SEL      RB1

          GETI     SEL15              ; STACK POINTER SET
          MOV      XA, #STACK         ;
          MOV      SP, XA             ;

          MOV      A, #0011B
          MOV      POC, A             ; PCC ← 0011B
```

```
; **       DATA RAM 0H-13FH ZERO CLEAR      **

           SEL       MB1
           MOV       HL, #3FH
           MOV       XA, #00
LOOP1:     MOV       @HL, A              ; 100H-13FH
           DECS      HL
           BR        LOOP1
           SEL       MB0
LOOP2 :    MOV       @HL, A              ; 0H-FFH
           DECS      HL
           BR        LOOP2

; **       TIMER SET (SAMPLING TIME = 30MSEC, FXX = 4.19MHz)  **

           GETI      SEL15               ; SEL      MB15
           MOV       XA, #79H
           MOV       TMOD0, XA
           MOV       XA, #01001100B
           MOV       TM0, XA
           EI
           EI        IET0
           SEL       MB1
LOOP3 :    MOV       XA, #0H
           MOV       B, #00H
LOOP4 :    SKE       B, #08H
           BR        LOOP4
           CALL      !HEIKIN
           MOV       TDATA, XA
           CALL      !SIOSUB
           BR        LOOP3

; ***      HEIKIN    (SAMPLE NUMBERS = 8)       ***
SEG3       CSEG      SENT
HEIKIN:    MOV       C, #2H
LOOP5 :    XCH       A, X
           CLR1      CY
           RORC      A
           XCH       A, X
           RORC      A
           DECS      C
           BR        LOOP5
           RET

           END
```

7.2 RELOCATION ATTRIBUTES AND INSTRUCTIONS

This section describes the relations between the relocation attributes and some instructions.

7.2.1 INBLOCK and INBLOCKA Attributes and Branch Instructions

In the case of the code segment with the INBLOCK attribute or the INBLOCKA attribute (referred to as INBLOCK [A] herein after) specified as the relocation attribute, take note of the position where 2-byte branch instruction "BRCB" is to be described as explained below.

The following four types of branch instructions are available for the 75X Series/75XL Series.

<1> 3-byte branch instruction "BRA !addr1" to the 16-bit absolute address

<2> 3-byte branch instruction "BR !addr" to the 14-bit absolute address

<3> 2-byte branch instruction "BRCB !caddr" to the inside of own block

<4> 1-byte branch instruction "BR $addr" to the 5-bit relative address

All branch instructions except <3> can be described anywhere in the program memory.

The <3> "BRCB !caddr" instruction can only be branched to the inside of the block indicated by the program counter at the point when this instruction is executed. If this instruction is located on the block boundary (XFFEH or XFFFH), the following unfavorable situation will result.

When the 75X Series/75XL Series executes the BRCB instruction, it checks the program counter value to determine the branch destination. However, the program counter indicates a point 2 bytes ahead of the BRCB instruction (that is, the next instruction to the BRCB instruction) at that point. In other words, the program counter points to the inside of the next block adjacent to the block where the BRCB instruction exists. Thus, the BRCB instruction branches to the inside of the next block instead of its own block.

If such a situation can be detected by the assembler, no problem will occur. However, it cannot be checked in the case of a relocatable code segment because the last location address is not determined in the assembly stage. The situation can only be checked in the linkage stage. For this reason, if the BRCB instruction is located on the block boundary, the linker will generate an error. If this error is overlooked, the program will not operate correctly.

It should be noted, however, that the BRCB instruction may only be located on the block boundary in the INBLOCK [A] attribute code segment when this instruction is described at the end of the code segment. This is because the INBLOCK [A] attribute code segment has a maximum possible length limited by the block size and the BRCB instruction not at the end of the segment cannot be located on the block boundary. Accordingly, the above problem can be avoided by taking extra care only when the BRCB instruction is to be described in the last statement in the code segment with the INBLOCK [A] relocation attribute.

Next, the 'Branch table auto creation function' of the linker is described.

The assembler optimizes the branch instruction auto select pseudo-instructions (referred to as the BR pseudo-instruction herein after) in the INBLOCK [A] attribute code segment as follows.

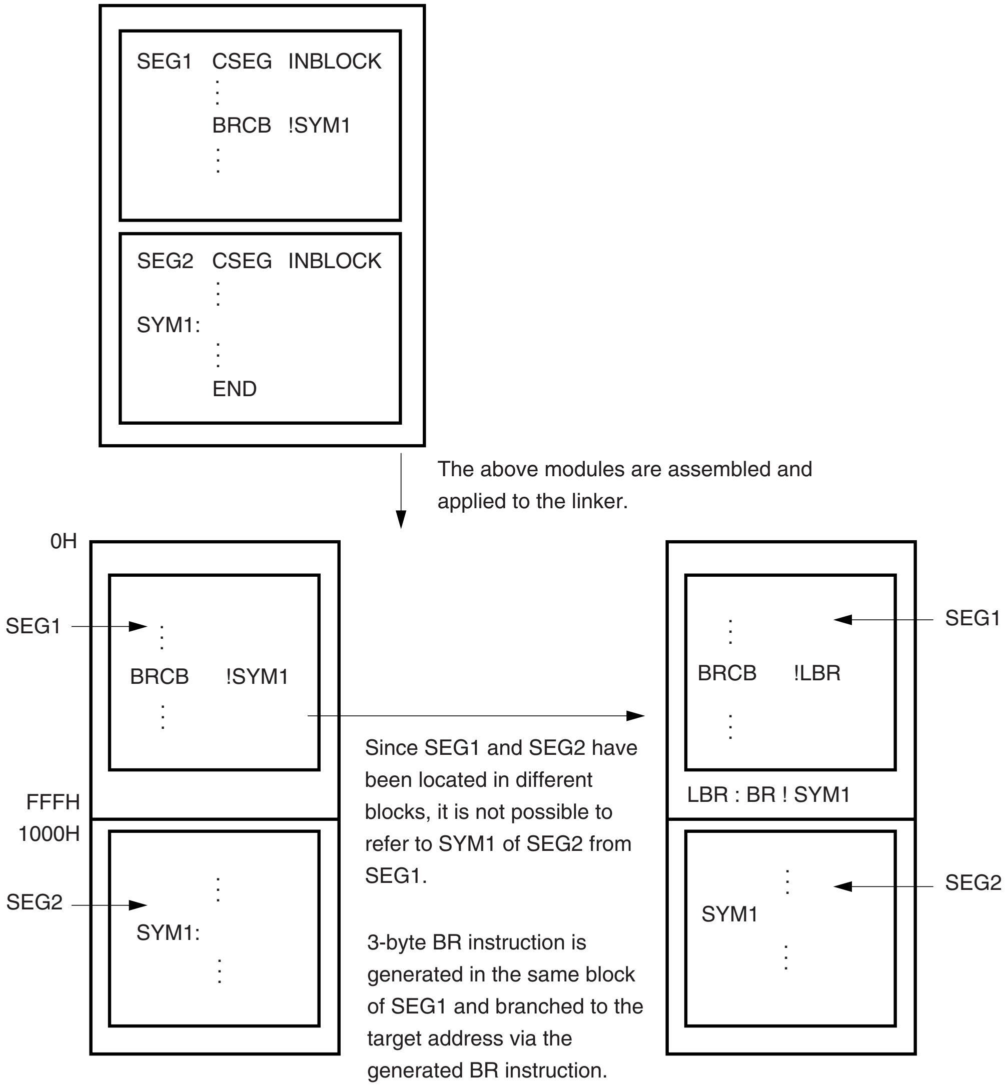
- Replaces the BR pseudo-instructions with 1-byte relative branch instructions if possible.
- Replaces all other BR pseudo-instructions with 2-byte BRCB instructions.

If the referred symbol is a relocatable symbol (including the external reference name) and the absolute address has not been determined, the 2-byte BRCB instruction code is generated as an object. If, in this case, the relocatable symbol is relocated in the block other than the one of the code segment which refers to the symbol, the symbol cannot be referred to by the BRCB instruction created by the assembler.

To solve that problem, the branch table auto creation function of the linker has been devised. In the above situation, the linker generates a 3-byte branch instruction in an empty area of the block where the code segment has been located as shown in **Figure 7-1. INBLOCK and INBLOCKA Attributes and Branch Instructions**. Namely, branch occurs by the BRCB branch instruction to the 3-byte branch instruction and then the code which refers to the virtual branchdestination symbol is created by the linker.

Refer to the Operation manual for details of the branch table auto creation function.

Figure 7-1. INBLOCK and INBLOCKA Attributes and Branch Instructions


SEG1 CSEG INBLOCK
:
BRCB !SYM1
:
SEG2 CSEG INBLOCK
:
SYM1:
:
END
The above modules are assembled and applied to the linker.
0H
SEG1
:
BRCB !SYM1
:
FFFH
1000H
SEG2
:
SYM1:
:
Since SEG1 and SEG2 have been located in different blocks, it is not possible to refer to SYM1 of SEG2 from SEG1.
3-byte BR instruction is generated in the same block of SEG1 and branched to the target address via the generated BR instruction.
:
BRCB !LBR
:
SEG1
LBR : BR ! SYM1
:
SYM1
:
SEG2

7.2.2 XBLOCK and XBLOCKA Attributes and Branch Instructions

The code segments with the XBLOCK attribute or the XBLOCKA attribute (referred to as XBLOCK [A] below) specified as the relocation attribute are relocated irrespective of the block. It means that those segments may be located on the block boundary.

Thus, 2-byte BRCB instruction cannot be described in the code segments having XBLOCK [A] attribute. If it is described, errors will result in the assembly stage.

The BR pseudo-instruction described in the XBLOCK [A] attribute code segment is optimized as follows:

- The BR pseudo-instructions are replaced with 1-byte relative branch instructions if possible.
- All other BR pseudo-instructions are replaced with 3-byte absolute branch instructions.

Accordingly, the branch table creation function of the linker has no meaning in the XBLOCK [A] attribute code segment.

7.2.3 Relocation Attributes and Subroutine Call Instructions

The following three types of subroutine call instructions are available for the 75X Series/75XL Series.

<1> 3-byte call instruction “CALLA !addr1” to the 16-bit absolute address
<2> 3-byte call instruction “CALL !addr” to the 14-bit absolute address
<3> 2-byte call instruction “CALLF !faddr” to the 11-bit absolute address.

In the case of <1>, the whole space of the program memory (with a maximum of 64 Kbytes) can be referred to. In the case of <2> or <3>, the maximum reference address range is limited as shown in **Table 7-1. Subroutine Call Instructions and Relocation Attributes**. Thus, when using a subroutine call in the case of <2> or <3>, the code segment referred to must be located in the address range shown in the table. For that purpose, the code segment including the reference destination entry address must have one of the relocation attributes listed in the table.

Table 7-1. Subroutine Call Instructions and Relocation Attributes

	Call Instruction	Reference Enable Address[Note]	Relocation Attribute of Reference Destination Code Segment
<1>	“CALLA !addr1”	0000H to FF7FH	INBLOCKA, XBLOCKA, INBLOCK, XBLOCK, SENT
<2>	“CALL !addr”	0000H to 3FFFH	INBLOCK, XBLOCK, SENT
<3>	“CALLF !faddr”	0000H to 07FFH	SENT

Note The last address described in the section relating to “CALLA !addr1” and “CALL !addr” instructions is the maximum value. The actual value will become smaller because the on-chip ROM capacity differs depending on the unit type.

7.2.4 IENT Attribute and GETI Instruction

The code segment with ‘IENT’ specified for the relocation attribute is located at addresses 20H to 7FH of the program memory.

The GETI instruction is available for the 75X Series/75XL Series. This instruction is used to refer to the 2-byte table in the program memory and to execute the following instructions with one byte. It considerably helps to decrease the program size.

- Two 1-byte instructions
- 2-byte instructions (except “BRCB !caddr” and “CALLF !faddr”)
- 3-byte instructions “BR !addr”, “CALL !addr”, “BRA !addr1” and “CALLA !addr1”

3-byte branch instructions and subroutine call instructions in particular can be executed by the GETI instruction efficiently.

The table referred to by the GETI instruction must be at addresses 20H to 7FH of the program memory. Thus, IENT is specified as the relocation attribute for the code segment for the table referred to by the GETI instruction.

When specifying an absolute address for the GETI instruction reference table code segment, an even address in the range of 20H to 7FH must be specified.

Where and how to locate the GETI instruction reference table code segment have now been described. Next, actual programming of the GETI instruction reference table is shown in Example that follows.

Example GETI instruction reference table code segment

```
;  TABLE FOR GETI
   EXAMPLE         CSEG       IENT                  ← (a)
;  ** TWO 1 BYTE INSTRUCTION **
   MOVAHL :        MOV        A, @HL              ┐
                   INCS       L                   ┘ ← (b)
   XCHADE :        XCH        A, @DE              ┐
                   INCS       DE                  ┘ ← (c)
;  ** 2 BYTE INSTRUCTION **
   SETFLAG :       SET1       FLAG                  ← (d)
;  ** 3 BYTE INSTRUCTION **
   CERR1 :         TCALL      ERROR1                ← (e)
   CERR2 :         TCALL      ERROR2                ← (f)
   BEXIT1 :        TBR        EXIT1                 ← (g)
   BEXIT2 :        TBR        EXIT2                 ← (h)
```

(a): The CSEG pseudo-instruction is used to instruct the assembler to start the code segment. Since this code segment is intended for the GETI instruction reference table, IENT is specified for the relocation attribute.

(b), (c): Two 1-byte instructions to be executed with one byte using the GETI instruction are described. When using these two 1-byte instructions in the program, describe the label described in the symbol column of the first 1-byte instruction as the GETI instruction operand as follow:

GETI MOVAHL or GETI XCHADE

(d): 2-byte instruction to be executed with one byte using the GETI instruction is described. When using this 2-byte instruction in the program, describe as follows:

GETI SETFLAG

(e), (f): The TCALL pseudo-instruction has been described.

(g), (h): The TBR pseudo-instruction has been described.

The TBR and TCALL pseudo-instructions are used to define the GETI instruction table.
For details of the TBR and TCALL pseudo-instructions, refer to **4.8 GETI INSTRUCTION TABLE DEFINITION PSEUDO-INSTRUCTIONS**.

7.2.5 PAGE Attributes and MOVT, BR PCDE, and BR PCXA Instructions

The code segment with ‘PAGE’ specified for the relocation attribute has its start address assigned on any page boundary (××00H) in the program memory. This attribute is used in combination with the INBLOCKA, XBLOCKA, INBLOCK, XBLOCK and SENT attributes (if only PAGE is described as the relocation attribute, the relocation attribute of the code segment will be INBLOCK PAGE). The concept of ‘PAGE’ has been derived from the following instruction restrictions.

The 75X series/75XL series is provided with the “MOVT” instruction used to refer to the program memory table data..pa The “MOVT” instruction is also used to set the DE or XA register contents in the least significant 8 bits of the program counter and to transfer the program memory contents addressed by the register contents to the XA register. In this case, the most significant 8 bits of the program counter remain unchanged and the program memory table is addressed by the DE or XA register contents. Thus, the table data on the own page where this instruction is located can be referred to but date reference beyond the page boundary is not possible.

Further, the 75X series/75XL series is equipped with the “BR PCDE” and “BR PCXA” instructions. These instructions are intended to determine the branch destination according to the received data. They set the DE or XA register contents in the least significant 8 bits of the program counter and branch to the program memory addressed by the register contents. In this case also, the most significant 8 bits of the program counter remain unchanged. Thus, they can branch to their own page where they are located but they cannot branch beyond the page boundary.

For these reasons, in the case of the “MOVT”, “BR PCDE” and “BR PCXA” instructions, the data and address referred to by each instruction and the instructions themselves are programmed to be within 256 bytes and PAGE is specified as the relocation attribute to align the start address of the code segment to the page boundary.

APPENDIX A LIST OF ASSEMBLED RELEVANT UNIT TYPES

The assembled unit types for the RA75X assembler package are shown below.

- IE-75001-R, IE-75000-R[Note 1] and EVAKIT-75X[Note 2]

Target Device	–C Option Specified Value	ROM Range	RAM Range	Usable Register Pair	Usable MBn	Usable RBn	VENTn Pseudo-Instruction		Type Usable by Register Indirect Addressing	BRA and CALLA Instructions
							Usable VENTn	n Value Usable for MBE = n and RBE = n		
μPD75000	000	0H to 3FFFH	0H to 0F7FH	XA, BC, DE, HL, XA', BC', DE', HL'	MB0 to MB15	RB0 to RB15	VENT0 to VENT5	BME : 0, 1 RBE : 0, 1	@BCDE, @BCXA, @PCDE, @PCXA, @HL+, @HL–, @DE, @DL, @HL, @H+mem.bit, pmem.@L	Impossible
μPD75000A	000A	0H to FF7FH	0H to 0F7FH							Possible

Notes 1. Maintenace product
(No longer available for purchase)

2. Discontinuation product
(No longer available for purchase)

- Expanded High-end device

Target Device	–C Option Specified Value	ROM Range	RAM Range	Usable Register Pair	Usable MBn	Usable RBn	VENTn Pseudo-Instruction		Type Usable by Register Indirect Addressing	BRA and CALLA Instructions
							Usable VENTn	n Value Usable for MBE = n and RBE = n		
μPD75117H μPD75P117H	117H	0H to 5F7FH	0H to 02FFH	XA, BC, DE, HL, XA', BC', DE', HL'	MB0 to MB2 MB15	RB0 to RB3	VENT0 to VENT5	MBE : 0, 1 RBE : 0, 1	@BCDE, @BCXA, @PCDE, @PCXA, @HL+, @HL–, @DE, @DL, @HL, @H+mem.bit, pmem.@L (μPD75217 only; except @BCDE and @BCXA)	Possible
μPD75217	217	0H to 5F7FH	0F to 02FFH				VENT0 to VENT7			
μPD75218 μPD75P218	218	0H to 7F7FH	0H to 03FFH		MB0 to MB3 MB15					
μPD75236	236	0H to 3F7FH	0H to 02FFH		MB0 to MB2, MB15					
μPD75237	237	0H to 5F7FH	0H to 03FFH		MB0 to MB3 MB15		VENT0 to VENT6			
μPD75238 μPD75P238	238	0H to 7F7FH	0H to 03FFH							
μPD75517	517	0H to 5F7FH	0H to 03FFH				VENT0 to VENT7			
μPD75518 μPD75P518	518	0H to 7F7FH	0H to 03FFH							
μPD75617A	617A	0H to 5F7FH	0H to 05FFH[Note]		MB0 to MB5, MB15					

Note 8-bit data transfer instructions (MOV XA, mem/MOV mem, XA/XCH XA, mem) cannot be used in the address range of 0100H to 0127H.

Phase-out/Discontinued

- High-end device

Target Device	–C Option Specified Value	ROM Range	RAM Range	Usable Register Pair	Usable MBn	Usable RBn	VENTn Pseudo-Instruction		Type Usable by Register Indirect Addressing	BRA and CALLA Instructions
							Usable VENTn	n Value Usable for MBE = n and RBE = n		
μPD75104 μPD75104A	104	0H to 0FFFH	0H to 013FH	XA, BC, DE, HL, XA', BC', DE', HL'	MB0 to MB1 MB15	RB0 to RB3	VENT0 to VENT5	MBE : 0, 1 RBE : 0, 1	@PCDE, @PCXA, @HL+, @HL–, @DE, @DL, @HL, @H+mem.bit, pmem.@L	Impossible
μPD75106	106	0H to 177FH	0H to 013FH							
μPD75108 μPD75108F μPD75108A μPD75P108B	108	0H to 1F7FH	0H to 01FFH							
μPD75P108	P108	0H to 1FFFH	0H to 01FFH							
μPD75112 μPD75112F	112	0H to 2F7FH	0H to 01FFH							
μPD75116 μPD75116F μPD75P116	116	0H to 3F7FH	0H to 01FFH							
μPD75116H	116H	0H to 3F7FH	0H to 02FFH		MB0 to MB2, MB15					
μPD75206	206	0H to 177FH	0H to 013FH[Note 1]		MB0 to MB5, MB15		VENT0 to VENT7			
μPD75208	208	0H to 1F7FH	0H to 01BFH[Note 1]							
μPD75CG208	CG208	0H to 1FFFH	0H to 01BFH[Note 1]							
μPD75212A	212A	0H to 2F7FH	0H to 01FFH							
μPD75216A μPD75P216A	216A	0H to 3F7FH	0H to 01FFH							
μPD75CG216A	CG216A	0H to 3FFFH	0H to 01FFH							
μPD75336 μPD75P336	336	0H to 3F7FH	0H to 02FFH[Note 2]		MB0 to MB2 MB15		VENT0 to VENT6			
μPD75352A	352A	0H to 2F7FH	0H to 03FFH[Note 3]		MB0 to MB3, MB15					
μPD75512	512	0H to 2F7FH	0H to 01FFH		MB0 to MB1 MB15					
μPD75516 μPD75P516	516	0H to 3F7FH	0H to 01FFH							

Notes **1.** The RAM range also includes the display memory. The display memory means a total of 49 nibbles of RAM address 1C0H to 1FFH except 1C3H, 1C7H, 1CBH, 1CFH, 1D3H, 1D7H, 1D8H, 1DFH, 1E3H, 1E7H, 1EBH, 1EFH, 1F3H, 1F7H and 1FBH.

2. 8-bit transfer instructions (MOV XA, mem/MOV mem, XA/ XCH XA, mem) cannot be used in the address rang eof 01E8H to 01FFH.

3. 8-bit trtansfer instructions (MOV XA, mem/MOV mem, XA/XCH XA, mem) cannot be used in the address range of 0100H to 0126H.

- Standard device

Target Device	–C Option Specified Value	ROM Range	RAM Range	Usable Register Pair	Usable MBn	Usable RBn	VENTn Pseudo-Instruction		Type Usable by Register Indirect Addressing	BRA and CALLA Instructions
							Usable VENTn	n Value Usable for MBE = n and RBE = n		
μPD75004	004	0H to 0FFFH	0H to 01FFH	XA, BC, DE, HL,	MB0 MB1 MB15	Not usable	VENT0 to VENT5	MBE : 0, 1 RBE : 0, (Fixed)	@PCDE, @PCXA, @DE, @DL, @HL, @H+mem.bit, pmem.@L	Impossible
μPD75006	006	0H to 177FH	0H to 01FFH							
μPD75008 μPD75P008	008	0H to 1F7FH	0H to 01FFH							
μPD75028	028	0H to 1F7FH	0H to 01FFH				VENT0 to VENT6			
μPD75036 μPD75P036	036	0H to 3F7FH	0H to 03FFH		MB0 to MB3 MB15					
μPD75048 μPD75P048	048	0H to 1F7FH	0H to 01FFH 400H to 07FFH[Note 1]		MB0 to MB7, MB15		VENT0 to VENT7			
μPD75064	064	0H to 0FFFH	0H to 01FFH		MB0 MB1 MB15		VENT0 to VENT5			
μPD75066	066	0H to 177FH	0H to 01FFH							
μPD75068 μPD75P068	068	0H to 1F7FH	0H to 01FFH							
μPD75268	268	0H to 1F7FH	0H to 01FFH							
μPD75304 μPD75304B	304	0H to 0FFFH	0H to 01FFH[Note 2]							
μPD75306 μPD75306B	306	0H to 177FH	0H to 01FFH[Note 2]							
μPD75308 μPD75308B μPD75P308	308	0H to 1F7FH	0H to 01FFH[Note 2]							
μPD75312	312	0H to 2F7FH	0H to 01FFH[Note 2]							
μPD75312B	312B	0H to 2F7FH	0H to 03FFH[Note 2]		MB0 to 3, MB15					
μPD75316 μPD75P316	316	0H to 3F7FH	0H to 01FFH[Note 2]		MB0, MB1, MB15					
μPD75P316A	316A	0H to 3F7FH	0H to 03FFH[Note 2]		MB0 to 3, MB15					
μPD75316B μPD75P316B	316B	0H to 3F7FH	0H to 03FFH[Note 2]							
μPD75328 μPD75P328	328	0H to 1F7FH	0H to 01FFH[Note 3]		MB0, MB1 MB15					

Notes
1. EEPROM is allocated in the addresses 0400H to 07FFH.
2. 8-bit transfer instructions (MOV XA, mem/MOV mem, XA/XCH XA, mem) cannot be used in the address range of 01E0H to 01FFH.
3. 8-bit transfer instructions (MOV, XA, mem/MOV mem, XA/XCH XA, mem) cannot be used in the address range of 01E8H to 01FFH.

- Low-end device

Target Device	–C Option Specified Value	ROM Range	RAM Range	Usable Register Pair	Usable MBn	Usable RBn	VENTn Pseudo-Instruction		Type Usable by Register Indirect Addressing	BRA and CALLA Instructions
							Usable VENTn	n Value Usable for MBE = n and RBE = n		
μPD75402A μPD75P402	402	0H to 077FH	0H to 003FH	XA, HL	Not usable	Not usable	VENT0 to VENT2 VENT4	MBE : 0 (Fixed) RBE : 0 (Fixed)	@PCXA, @HL	Impossible

★

- 75XL Series device

(1/2)

Target Device	–C Option Specified Value	ROM Range	RAM Range	Usable Register Pair	Usable MBn	Usable RBn	VENTn Pseudo-Instruction		Type Usable by Register Indirect Addressing	BRA and CALLA Instructions
							Usable VENTn	n Value Usable for MBE = n and RBE = n		
μPD750004	0004	0H to 0FFFH	0H to 1FFH	XA, BC, DE, HL, XA', BC', DE', HL'	MB0, MB1, MB15	RB0 to RB3	VENT0 to VENT6	MBE : 0, 1 RBE : 0, 1	@BCDE, @BCXA, @PCDE, @PCXA, @HL+, @HL–, @DE, @DL, @HL, @H+mem.bit, pmem.@L	Possible in MkII mode only
μPD750006	0006	0H to 17FFH								
μPD750008	0008	0H to 1FFFH								
μPD75P0016	P0016	0H to 3FFFH								
μPD750104	0104	0H to 0FFFH								
μPD750106	0106	0H to 17FFH								
μPD750108	0108	0H to 1FFFH								
μPD75P0116	P0116	0H to 3FFFH								
μPD750064	0064	0H to 0FFFH								
μPD750066	0066	0H to 17FFH								
μPD750068	0068	0H to 1FFFH								
μPD75P0076	P0076	0H to 3FFFH								
μPD753012	3012	0H to 2FFFH	0H to 3FFH[Note]		MB0 to MB3, MB15					
μPD753016	3016	0H to 3FFFH								
μPD753017	3017	0H to 5FFFH								
μPD75P3018	P3018	0H to 7FFFH								
μPD753012A	3012A	0H to 2FFFH								
μPD753016A	3016A	0H to 3FFFH								
μPD753017A	3017A	0H to 5FFFH								
μPD75P3018A	P3018A	0H to 7FFFH								

Note The display memory is allocated in addresses 1E0H to 1FFH.

★

- 75XL Series device

(2/2)

Target Device	–C Option Specified Value	ROM Range	RAM Range	Usable Register Pair	Usable MBn	Usable RBn	VENTn Pseudo-Instruction		Type Usable by Register Indirect Addressing	BRA and CALLA Instructions
							Usable VENTn	n Value Usable for MBE = n and RBE = n		
μPD753036	3036	0H to 3FFFH	0H to 2FFH[Note 2]	XA, BC, DE, HL, XA', BC', DE', HL'	MB0 to MB2, MB15	RB0 to RB3	VENT0 to VENT6	MBE : 0, 1 RBE : 0, 1	@BCDE, @BCXA, @PCDE, @PCXA, @HL+, @HL–, @DE, @DL, @HL, @H+mem.bit, pmem.@L	Possible in MkII mode only
μPD75P3036	P3036									
μPD753104	3104	0H to 0FFFH	0H to 1FFH[Note 3]		MB0, MB1, MB15					
μPD753106	3106	0H to 17FFH								
μPD753108	3108	0H to 1FFFH								
μPD75P3116	P3116	0H to 3FFFH								
μPD753204	3204	0H to 0FFFH	0H to 1FFH[Note 4]				VENT0 to VENT2, VENT4 to VENT6			
μPD753206	3206	0H to 17FFH								
μPD753208	3208	0H to 1FFFH								
μPD75P3216	P3216	0H to 3FFFH								
μPD753304[Note 1]	3304	0H to 0FFFH	0H to 0FFH 1E0H to 1F7H[Note 5]				VENT0, VENT1, VENT3, VENT5			
μPD754202	4202	0H to 07FFH	0H to 07FH		MB0, MB15		VENT0 to VENT2, VENT5, VENT6			
μPD754144	4144	0H to 0FFFH	0H to 07FH, 0400H to 041FH[Note 6]		MB0, MB4, MB15		VENT0 to VENT2, VENT5 to VENT7			
μPD754244	4244									
μPD754264	4264									
μPD75F4264[Note 1]	F4264									
μPD754302	4302	0H to 07FFH	0H to 0FFH		MB0, MB15		VENT0 to VENT6			
μPD754304	4304	0H to 0FFFH								
μPD75P4308	P4308	0H to 1FFFH								

Notes 1. Under development
2. The display memory is allocated in addresses 1ECH to 1FFH.
3. The display memory is allocated in addresses 1E0H to 1F7H.
4. The display memory is allocated in addresses 1ECH to 1F7H.
5. The display memory is allocated in addresses 1E0H to 1F7H.
6. EEPROM is allocated in addresses 0400H to 041FH.

[MEMO]

APPENDIX B LIST OF RESERVED WORDS

This is a compilation of the reserved words of the assembler package.
It should be of help during program development.

★ There are six types of reserved words. They are the machine code instruction, control instruction, pseudo-instruction, operator, register name, and specific address name code. The reserved words are the character strings which the assembler has reserved and cannot be put to use for unspecified purposes.

The types of reserved words describable in each column of the source program and a list of reserved words are shown below.

Symbol column	None of the reserved words can be described.
Mnemonic column	Only machine code instructions and pseudo-instructions can be described.
Operand column	Only the operators, register names and specific address name codes can be described.
Comment column	All reserved words can be described.

On the following pages, the symbols for the devices in the reserved word list, indicate target devices in the table below.

Symbol	Target Device
EV1	μPD75000
EV2	μPD75000A
0××	μPD75004, 75006, 75008, 75P008
02×	μPD75028, 75036, 75P036
04×	μPD75048, 75P048
06×	μPD75064, 75066, 75068, 75P068
1××	μPD75104, 75104A, 75106, 75108, 75108F, 75108A, 75P108, 75P108B, 75112, 75112F, 75116, 75116F, 75P116
116H	μPD75116H
117H	μPD75117H, 75P117H
2××	μPD75206, 75208, 75CG208, 75212A, 75216A, 75P216A, 75CG216A
217	μPD75217
218	μPD75218, 75P218
237	μPD75236, 75237, 75238, 75P238
26×	μPD75268
3××	μPD75304, 75306, 75308, 75304B, 75306B, 75308B, 75312, 75316, 75312B, 75316B, 75P308, 75P316, 75P316A, 75P316B
32×	μPD75328, 75P328
336	μPD75336, 75P336
34×	μPD75352A
4××	μPD75P402, 75402A
5××	μPD75512, 75516, 75P516
517	μPD75517, 75518, 75P518
6××	μPD75617A
75XL	75XL Series device[Note]

★ **Note** For specific address name code, refer to the Before Using a Device File that comes with the device file purchased.

List of Reserved Words

Reserved Word		Unit Type																						
		EV1	EV2	0××	02×	04×	06×	1××	116H	117H	2××	217	218	237	26×	3××	32×	336	34×	4××	5××	517	6××	75XL
Instruction	ADDC	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ADDS	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	AND	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	AND1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	BR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	BRA	×	○	×	×	×	×	×	○	○	×	○	○	○	×	×	×	×	×	×	×	○	○	○
	BRCB	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	BRK	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	CALL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	CALLA	×	○	×	×	×	×	×	○	○	×	○	○	○	×	×	×	×	×	×	×	○	○	○
	CALLF	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	CLR1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DECS	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DI	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	EI	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	GETI	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	HALT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	IN	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b	a
	INCS	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	MOV	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	MOV1	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	○
	MOVT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOP	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOT1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	OR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	OR1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	OUT	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b	a
	OUT3	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	POP	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	PUSH	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

Remarks 1. ○ : Applied
× : Not applied
2. a : PORT0 to 15
b : PORT0 to 20

Reserved Word		Unit Type																						
		EV1	EV2	0××	02×	04×	06×	1××	116H	117H	2××	217	218	237	26×	3××	32×	336	34×	4××	5××	517	6××	75XL
Instruction	RET	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	RETI	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	RETS	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ROLC	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	RORC	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SEL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	SET1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	SKE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SKF	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SKT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	SKTCLR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	STOP	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SUBC	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	SUBS	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	XCH	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	XOR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	XOR1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Operator	AND	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	EQ	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	GE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	GT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	HIGH	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	LE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	LOW	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	LT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	MOD	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	OR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SHL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SHR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	XOR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Control instruction	BREAK	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	CASE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	CAP CA	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	CONDITION COND	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

Phase-out/Discontinued

Reserved Word		Unit Type																						
		EV1	EV2	0××	02×	04×	06×	1××	116H	117H	2××	217	218	237	26×	3××	32×	336	34×	4××	5××	517	6××	75XL
Control instruction	CONTINUE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DEBUG DG	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DEBUGA DA	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DEFAULT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	EJECT EJ	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ELSE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ELSEIF	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ELSEIF_BIT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ENDIF	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ENDS	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ENDW	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ERRLOG EL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	FOR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	GENERATE GEN	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	GOTO	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	IF	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	IF_BIT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	IFCHR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	IFDEF	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	INCLUDE IC	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	LIST LI	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	LODM LM	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	MODE MD	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	MSGLOG ML	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NEXT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOCAP NOCA	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOCONDITION NOCOND	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

Reserved Word		Unit Type																						
		EV1	EV2	0××	02×	04×	06×	1××	116H	117H	2××	217	218	237	26×	3××	32×	336	34×	4××	5××	517	6××	75XL
Control instruction	NODEBUG NODB	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NODEBUGA NODA	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOGENERATE NOGEN	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOLIST NOLI	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOSYMBOLS NOSB	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOSYMLEN NOSL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NOXREF NOXR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	PAGELENGTH PL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	PAGEWIDTH	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	PWTAB	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	REPEAT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SB	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SWITCH	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SYMBOLS	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SYMLEN	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	TAB TB	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	TITLE TT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	UNTIL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	UNTIL_BIT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	WHILE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	WHILE_BIT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	XREF XF	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Other control instruction	DGL[Note]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DGS[Note]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	TOL_INF[Note]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

Note Special control instructions output from the structured assembler preprocessor.

Reserved Word		Unit Type																						
		E V 1	E V 2	0 × ×	0 2 ×	0 4 ×	0 6 ×	1 × ×	1 1 6 H	1 1 7 H	2 × ×	2 1 7	2 1 8	2 3 7	2 6 ×	3 × ×	3 2 ×	3 3 6	3 4 ×	4 × ×	5 × ×	5 1 7	6 × ×	7 5 X L
Pseudo-instruction	BR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	CSEG	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DB	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DBIT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DS	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DSEG	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	DW	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○
	END	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	EQU	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	EXTRN	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	NAME	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ORG	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	PUBLIC	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	SET	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	STKLN	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	TBR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	TCALL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	VENTn[Note]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Macros	MACRO	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	ENDM	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	EXITM	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	REPT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	IRP	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	IRPC	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	GLOBAL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Special	$	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	STACK	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Register	A	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	B	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	BC	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	BC'	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	○
	BCDE	○	○	×	×	×	×	×	○	○	×	×	○	○	×	×	×	×	×	×	×	○	○	○
	@BCDE	○	○	×	×	×	×	×	○	○	×	×	○	○	×	×	×	×	×	×	×	○	○	○
	BCXA	○	○	×	×	×	×	×	○	○	×	×	○	○	×	×	×	×	×	×	×	○	○	○
	@BCXA	○	○	×	×	×	×	×	○	○	×	×	○	○	×	×	×	×	×	×	×	○	○	○

Note n = 0 to 7 (depending on the device)

Reserved Word		Unit Type																						
		EV1	EV2	0××	02×	04×	06×	1××	116H	117H	2××	217	218	237	26×	3××	32×	336	34×	4××	5××	517	6××	75XL
Register	C	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	D	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	DE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	DE'	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	○
	@DE	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	@DL	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	E	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	H	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	HL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	HL'	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	○
	HL+	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	HL–	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	@HL	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	@HL+	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	○
	@HL–	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	○
	L	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	MBn	a	a	b	b	b	b	b	d	d	b	c	d	d	b	b	b	b	b	×	b	d	d	d
	PCDE	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	○
	@PCDE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○
	PCXA	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	○
	@PCXA	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	RBn	a	a	×	×	×	×	e	e	e	e	e	e	e	×	×	×	e	e	×	e	e	e	e
	X	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	XA	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	XA'	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	○
Specific address name code	ACKD	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	ACKE	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	ACKT	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	ADM	○	○	×	○	○	○	×	×	×	×	×	×	○	×	×	○	○	○	×	○	○	○	Note
	BP0	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	×	×	×	×	×	Note
	BP1	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	×	×	×	×	×	Note
	BP2	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	×	×	×	×	×	Note

★ **Note** For specific address name code, refer to the Before Using a Device File that comes with the device file purchased.

Remark	a: 0 to 15	c: 0 to 2, 15	e: 0 to 3
	b: 0, 1, 15	d: 0 to 3, 15	f: 0 to 7, 15

Phase-out/Discontinued

Reserved Word		Unit Type																						
		E V 1	E V 2	0 × ×	0 2 ×	0 4 ×	0 6 ×	1 × ×	1 1 6 H	1 1 7 H	2 × ×	2 1 7	2 1 8	2 3 7	2 6 ×	3 × ×	3 2 ×	3 3 6	3 4 ×	4 × ×	5 × ×	5 1 7	6 × ×	7 5 X L
Specific address name code	BP3	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	×	×	×	×	×	Note
	BP4	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	×	×	×	×	×	Note
	BP5	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	×	×	×	×	×	Note
	BP6	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	×	×	×	×	×	Note
	BP7	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	×	×	×	×	×	Note
	BS	×	○	×	×	×	×	×	○	○	×	×	○	○	×	×	×	×	○	×	×	○	○	Note
	BSB0	○	○	○	○	○	○	○	×	×	×	×	×	○	×	○	○	○	○	×	○	○	○	Note
	BSB1	○	○	○	○	○	○	○	×	×	×	×	×	○	×	○	○	○	○	×	○	○	○	Note
	BSB2	○	○	○	○	○	○	○	×	×	×	×	×	○	×	○	○	○	○	×	○	○	○	Note
	BSB3	○	○	○	○	○	○	○	×	×	×	×	×	○	×	○	○	○	○	×	○	○	○	Note
	BSYE	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	BT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	BTM	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	CLOM	○	○	○	○	○	○	○	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	CMDD	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	CMDT	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	COI	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	CSIE	○	○	○	○	○	○	×	×	×	×	×	×	×	×	○	○	○	○	○	×	×	○	Note
	CSIE0	○	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	○	○	×	Note
	CSIE1	○	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	○	○	×	Note
	CSIM	○	○	○	○	○	○	×	×	×	×	×	×	×	×	○	○	○	○	○	×	×	○	Note
	CSIM0	○	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	○	○	×	Note
	CSIM1	○	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	○	○	×	Note
	DACE0	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	DACE1	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	DACS0	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	DACS1	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	DIGS	○	○	×	×	×	×	×	○	○	○	○	○	○	○	×	×	×	×	×	×	×	×	Note
	DIMS	○	○	×	×	×	×	×	○	○	○	○	○	○	○	×	×	×	×	×	×	×	×	Note
	DSPM	○	○	×	×	×	×	×	○	○	○	○	○	○	○	×	×	×	×	×	×	×	×	Note
	EOC	○	○	×	○	○	○	×	×	×	×	×	×	○	×	×	○	○	○	×	○	○	○	Note
	EOT	○	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	○	○	×	Note
	EWC	×	○	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	EWE	×	○	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	EWP	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	EWST	×	○	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note

★ **Note** For specific address name code, refer to the Before Using a Device File that comes with the device file purchased.

Reserved Word		Unit Type																						
		E V 1	E V 2	0 × ×	0 2 ×	0 4 ×	0 6 ×	1 × ×	1 1 6 H	1 1 7 H	2 × ×	2 1 7	2 1 8	2 3 7	2 6 ×	3 × ×	3 2 ×	3 3 6	3 4 ×	4 × ×	5 × ×	5 1 7	6 × ×	7 5 X L
Specific address name code	GATEC	×	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	×	×	×	Note
	IE0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	IE1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IE2	○	○	○	○	○	○	○	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	IE3	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IE4	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IEBT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	×	Note
	IEBWT	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	○	Note
	IECSI	○	○	○	○	○	○	×	×	×	×	×	×	×	×	○	○	○	○	○	×	×	○	Note
	IECSI0	○	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	○	○	×	Note
	IEEE	×	○	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IEKS	○	○	×	×	×	×	×	○	○	○	○	○	○	○	×	×	×	×	×	×	×	×	Note
	IEMFT	×	○	×	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IEMT0	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IEMT1	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IEOW	×	○	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IESIO	○	○	×	×	×	×	○	○	○	○	○	○	×	○	×	×	×	×	×	×	×	×	Note
	IET0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IET1	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	○	○	×	×	×	○	Note
	IETPG	○	○	×	×	×	×	×	○	○	○	○	○	○	×	×	×	×	×	×	○	○	×	Note
	IEW	○	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IM0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	IM1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IM2	○	○	○	○	○	×	×	×	×	×	×	×	×	×	○	○	○	○	×	○	○	○	Note
	IPS	○	○	○	○	○	×	○	○	○	○	○	○	○	×	○	○	○	○	○	○	○	○	Note
	IRQ0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	IRQ1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IRQ2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	IRQ3	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IRQ4	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IRQBT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	×	Note
	IRQBWT	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	○	Note
	IRQCSI	○	○	○	○	○	○	×	×	×	×	×	×	×	×	○	○	○	○	○	×	×	○	Note
	IRQCSI0	○	○	×	×	×	×	×	×	×	×	×	○	○	×	×	×	×	×	×	○	○	×	Note
	IRQEE	×	○	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IRQKS	○	○	×	×	×	×	×	○	○	○	○	○	○	○	×	×	×	×	×	×	×	×	Note

★ **Note** For specific address name code, refer to the Before Using a Device File that comes with the device file purchased.

Reserved Word		Unit Type																						
		EV1	EV2	0××	02×	04×	06×	1××	116H	117H	2××	217	218	237	26×	3××	32×	336	34×	4××	5××	517	6××	75XL
Specific address name code	IRQMFT	×	○	×	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IRQMT0	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IRQMT1	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IRQOW	×	○	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	IRQSIO	○	○	×	×	×	×	○	○	○	○	○	○	×	○	×	×	×	×	×	×	×	×	Note
	IRQT0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IRQT1	○	○	×	×	×	×	○	×	×	×	×	×	○	×	×	×	○	○	×	×	×	○	Note
	IRQTPG	○	○	×	×	×	×	×	○	○	○	○	○	○	×	×	×	×	×	×	○	○	×	Note
	IRQW	○	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IST0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	IST1	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	Note
	KR0	○	○	○	○	○	○	×	×	×	×	×	×	×	×	○	○	○	○	×	○	○	○	Note
	KR1	○	○	○	○	○	○	×	×	×	×	×	×	×	×	○	○	○	○	×	○	○	○	Note
	KR2	○	○	○	○	○	○	×	×	×	×	×	×	×	×	○	○	○	○	×	○	○	○	Note
	KR3	○	○	○	○	○	○	×	×	×	×	×	×	×	×	○	○	○	○	×	○	○	○	Note
	KR4	○	○	○	○	○	×	×	×	×	×	×	×	×	×	○	○	○	○	×	○	○	○	Note
	KR5	○	○	○	○	○	×	×	×	×	×	×	×	×	×	○	○	○	○	×	○	○	○	Note
	KR6	○	○	○	○	○	×	×	×	×	×	×	×	×	×	○	○	○	○	×	○	○	○	Note
	KR7	○	○	○	○	○	×	×	×	×	×	×	×	×	×	○	○	○	○	×	○	○	○	Note
	KS0	○	○	×	×	×	×	×	○	○	○	○	○	○	○	×	×	×	×	×	×	×	×	Note
	KS1	○	○	×	×	×	×	×	○	○	○	○	○	○	○	×	×	×	×	×	×	×	×	Note
	KS2	×	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	×	×	×	Note
	KSF	○	○	×	×	×	×	×	○	○	○	○	○	○	○	×	×	×	×	×	×	×	×	Note
	LCDC	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	○	×	×	×	○	Note
	LCDM	○	○	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	○	×	×	×	○	Note
	LPS	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	○	Note
	MBE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	MBS	×	○	×	×	×	×	×	○	○	×	×	○	○	×	×	×	×	○	×	×	○	○	Note
	MFTC	×	○	×	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MFTH	×	○	×	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MFTL	×	○	×	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MFTM	×	○	×	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MODH	○	○	×	×	×	×	×	○	○	○	○	○	○	×	×	×	×	×	×	○	○	×	Note
	MODL	○	○	×	×	×	×	×	○	○	○	○	○	○	×	×	×	×	×	×	○	○	×	Note
	MT0	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MT1	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note

★ **Note** For specific address name code, refer to the Before Using a Device File that comes with the device file purchased.

Reserved Word		Unit Type																						
		EV1	EV2	0××	02×	04×	06×	1××	116H	117H	2××	217	218	237	26×	3××	32×	336	34×	4××	5××	517	6××	75XL
Specific address name code	MTM0	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MTM1	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MTOE0	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MTOE1	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MTOF0	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	MTOF1	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	PCC	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	PDGB	×	○	×	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	PMGA	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	PMGB	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	PMGC	○	○	○	○	○	×	○	×	×	×	×	×	×	×	×	○	○	○	×	○	○	○	Note
	PMGD	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	○	Note
	PMGE	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	POGA	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	POGB	○	○	○	○	○	×	×	×	×	×	×	×	×	×	×	○	○	○	×	×	×	○	Note
	POGC	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	○	Note
	POGD	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	PONF	○	○	×	×	×	×	○	○	○	○	○	○	×	×	×	×	×	×	×	×	×	×	Note
	PORT0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	PORT1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	PORT2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	PORT3	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	PORT4	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	PORT5	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	PORT6	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	PORT7	○	○	○	○	○	×	○	×	×	×	×	×	○	×	○	○	○	○	×	○	○	○	Note
	PORT8	○	○	○	○	○	×	○	×	×	×	×	×	○	×	×	○	○	○	×	○	○	○	Note
	PORT9	○	○	×	○	○	×	○	×	×	×	×	×	○	×	×	×	×	○	×	○	○	○	Note
	PORT10	○	○	×	○	○	×	×	×	×	×	×	×	○	×	×	×	×	○	×	○	○	○	Note
	PORT11	○	○	×	○	○	○	×	×	×	×	×	×	○	×	×	×	×	○	×	○	○	○	Note
	PORT12	○	○	×	×	×	×	○	×	×	×	×	×	○	×	×	×	×	○	×	○	○	○	Note
	PORT13	○	○	×	×	×	×	○	×	×	×	×	×	○	×	×	×	×	○	×	○	○	○	Note
	PORT14	○	○	×	×	×	×	○	×	×	×	×	×	○	×	×	×	×	×	×	○	○	○	Note
	PORT15	○	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	○	○	○	Note
	PORT16	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	PORT17	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note

★ **Note** For specific address name code, refer to the Before Using a Device File that comes with the device file purchased.

Reserved Word		Unit Type																						
		E V 1	E V 2	0 × ×	0 2 ×	0 4 ×	0 6 ×	1 × ×	1 1 6 H	1 1 7 H	2 × ×	2 1 7	2 1 8	2 3 7	2 6 ×	3 × ×	3 2 ×	3 3 6	3 4 ×	4 × ×	5 × ×	5 1 7	6 × ×	7 5 X L
Specific address name code	PORT18	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	PORT19	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	PORT20	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	Note
	PORTH	○	○	×	×	×	×	×	○	○	○	○	○	○	○	×	×	×	×	×	×	×	×	Note
	PSW	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	PTH0	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	PTH1	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	PTHM	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	RBE	○	○	×	×	×	×	○	○	○	○	○	○	○	×	×	×	○	○	×	○	○	○	Note
	RBS	×	○	×	×	×	×	×	○	○	×	×	○	○	×	×	×	○	○	×	×	○	○	Note
	RELD	○	○	○	○	○	○	×	×	×	×	×	×	○	○	○	○	○	○	○	○	○	○	Note
	RELOAD	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	RELT	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	SA	○	○	×	○	○	○	×	×	×	×	×	×	○	×	×	○	○	×	×	○	○	○	Note
	SBS	×	○	×	×	×	×	×	○	○	×	○	○	○	×	×	×	×	×	×	×	○	○	Note
	SCC	○	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	SEGEX	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	SIO	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○	○	○	×	×	○	Note
	SIO0	○	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	○	○	×	Note
	SIO1	○	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	○	○	×	Note
	SIOM	○	○	×	×	×	×	○	○	○	○	○	○	×	○	×	×	×	×	×	×	×	×	Note
	SOC	○	○	×	○	○	○	×	×	×	×	×	×	○	×	×	○	○	○	×	○	○	○	Note
	SP	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	STATA	×	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	×	×	×	Note
	STATB	×	○	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×	×	×	×	Note
	SVA	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note
	T0	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	T1	○	○	×	×	×	×	○	×	×	×	×	×	○	×	×	×	○	○	×	×	×	○	Note
	TBC0	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	TBC4	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	TBC8	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	TBCM	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	TGC	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	○	Note
	TGM	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	○	Note
	TGS	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	○	Note
	TI0	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note

★ **Note** For specific address name code, refer to the Before Using a Device File that comes with the device file purchased.

Reserved Word		Unit Type																						
		EV1	EV2	0××	02×	04×	06×	1××	116H	117H	2××	217	218	237	26×	3××	32×	336	34×	4××	5××	517	6××	75XL
Specific address name code	TI1	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	TM0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	TM1	○	○	×	×	×	×	○	×	×	×	×	×	○	×	×	×	○	○	×	×	×	○	Note
	TMOD0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	Note
	TMOD1	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	○	○	×	×	×	○	Note
	TO0	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	TO1	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	TOE0	○	○	○	○	○	○	○	×	×	×	×	×	○	×	○	○	○	○	×	○	×	○	Note
	TOE1	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	○	○	×	×	×	○	Note
	TOF0	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	TOF1	○	○	×	×	×	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	Note
	TPGM	○	○	×	×	×	×	×	○	○	○	○	○	○	×	×	×	×	×	×	○	○	×	Note
	WDTM	×	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	○	Note
	WM	○	○	○	○	○	○	×	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	Note
	WUP	○	○	○	○	○	○	×	×	×	×	×	×	○	×	○	○	○	○	○	○	○	○	Note

★ **Note** For specific address name code, refer to the Before Using a Device File that comes with the device file purchased.

APPENDIX C LIST OF PSEUDO-INSTRUCTIONS

This is a compilation of all of the pseudo-instructions of the assembler package.
It should be of help during program development.

Pseudo-Instruction			Function and Classification	Remarks
Symbol Column	Mnemonic Column	Operand Column		
Segment name	CSEG	[Relocation attribute specification]	Code segment start declaration	Segment name: Symbol Refer to Table 4-3 for symbol relocation attributes
Segment name	DSEG	[Bank value] [AT absolute formula]	Data segment start declaration	Segment name: Symbol
	ORG	Absolute formula	Location counter modification	Symbol forward reference disabled in operand formula
	NAME	Module name	Module name definition	Module name: Symbol
	PUBLIC	Symbol [, ...]	External definition name declaration	
	EXTRN	Type (symbol [, ...]) [, ...]	External reference name declaration	5 types: CODE, DATA, BIT, PBIT, NUMBER
Name	EQU	Formula	Name definition	Name: Symbol Symbol forward reference disabled in operand formula External reference name reference disabled
Name	SET	Formula	Redefinable name definition	Name: Symbol Symbol forward reference disabled in operand formula External reference name reference disabled
[Label:]	DB	$\left\{ \begin{matrix} \text{Formula} \\ \text{Character string} \end{matrix} \right\}$ [, ...]	Byte	Label: Symbol
[Label:]	DS	Absolute formula	Byte Data area secure	Label: Symbol
	STKLN	Absolute formula	Stack area secure	Symbol forward reference disabled in operand formula
[Label:]	BR	Formula	Branch instruction auto selection	Label: Symbol
	VENTn	$\text{MBE} = \left\{ \begin{matrix} 0, \\ 1 \end{matrix} \right\}$ $\text{RBE} = \left\{ \begin{matrix} 0, \\ 1 \end{matrix} \right\}$ Start address	Entry address area secure	Describe at the beginning of the source (before description of segment pseudo-instruction). Specify RBE = 0 when the assembled unit type is the 75X standard and MBE = 0 or RBE = 0 when the assembled unit type is the 75X low-end.
[Label:]	TCALL	Formula	GETI instruction Table creation	For CALL instruction Describable only in CSEG IENT attribute or even absolute address in the range 20H to 7FH
[Label:]	TBR	Formula	GETI instruction Table creation	For branch instruction Same as above
	END		End of source module	

[MEMO]

APPENDIX D LIST OF MAXIMUM PERFORMANCE CAPABILITIES

The maximum performance of assembler package is indicated for the following items.

- Source statement length
- No. of describable symbols
- No. of describable segments
- No. of branch tables which can be created
- ★ Other

(1) Source Statement Length

Program Name	Maximum Performance
Assembler	220 characters (including CR, LF)

(2) No. of Describable Symbols

Program Name	Maximum Performance	
★ Assembler	• In assembly	Approx. 3000
Linker	• Local symbol • External definition (PUBLIC) symbol • External reference (EXTRN) symbol	No limit Approx. 3000/all modules Approx. 500/module

(3) No. of Describable Segments

Program Name	Maximum Performance
Assembler	(a) to (c) total to approx. 120/module (a) No. of segment definition pseudo-instructions (b) No. of ORG pseudo-instructions (c) No. of VENT pseudo-instructions × 2
Linker	(a) to (d) total to approx. 250/all modules (a) No. of input modules × 2 (b) No. of segments (c) No. of ORG pseudo-instructions (d) No. of VENT pseudo-instructions × 2

(4) No. of Branch Tables Which can be Created

Program Name	Maximum Performance
Linker	Approx. 1000

★ (5) Other

Program Name	Maximum Performance	
Assembler	• No. of local symbols in one macro	100 (including tentative parameters)
	• Nest level	Approx. 64 Kbytes
	• Macro body field size	32 levels
	(including areas for the macro instruction, $IF, $SWITCH, and $INCLUDE instructions)	
	• Maximum repeat number of repeated macro	1023 times

APPENDIX E LIST OF PRECAUTIONS

Describes the precautions when the assemble package is used.

(1) Caution on memory bit manipulations

If immediate data is specified in the range 0FB0H.0 to 0FBFH.3 or 0FF0H.0 to 0FFFH.3, fmem.bit object code is generated.

○ Remedy

If you want to generate fmem.bit object code, be sure to specify a reserved word in the above range.

○ Reference

3.5 OPERAND CHARACTERISTICS

(2) Caution on segments with the same name

If there are segments with the same name in a single source module, list conversion may not be performed correctly.

○ Remedy

When the list converter is used, give different names to all segments in a single source module.

○ Reference

4.2 SEGMENT DEFINITION PSEUDO-INSTRUCTIONS

(3) Caution on source program writing

If a source program assembly list which does not obey rules <1> to <4> below is input, the list converter may abort with an error or the list may not be converted correctly.

<1> VENTn and ORG pseudo-instructions must be written in upper-case characters starting in column 9.
<2> The NOLIST control instruction must not be used.
<3> Identical segments must not be written in the same module.
<4> A segment definition pseudo-instruction must be written before an instruction that generates object code is written.

○ Reference

CHAPTER 4 PSEUDO-INSTRUCTIONS

(4) Caution on input files

There must be no errors in the following files input to the list converter.

● Assembly list file (.PRN)
● Object module file (.REL)
● Load module file (.LNK)

○ Reference

Operation Volume, 8.1 LIST CONVERTER INPUT/OUTPUT FILES

(5) Caution on input file names used by debugger

The first character of an input file name used by an in-circuit emulator or other debugger must not be a numeral. If a file of this kind is input, an error will result on the debugger side at load time.

○ Remedies

<1> Use a non-numeric character as the first character of a file name.
<2> Change existing file names with "name pseudo-instruction".

○ Reference

3.3.3 Character Configuration Fields
4.3 (1) NAME (name)

(6) Cautions on BRCB instruction (bugs)

<1> If the BRCB instruction jump destination address is a BLOCK external reference of the form "label number - constant", output will not be performed in ascending branch table map address order.

<2> If the BRCB instruction jump destination address is a BLOCK external reference of the form "label number - constant", and there is a BRCB instruction for which the description format of the same jump destination address differs in the same block, an extra branch table will be created.

○ Reference

3.4.1 (1) (b) – (subtraction)

(7) Caution on byte separation operators (HIGH, LOW)

If the term is a relocatable term or external reference term, nesting is not possible. If used in combination with a BRCB, EQU or SET instruction, only an absolute term can be used.

○ References

Table 3-10. Combination of Terms and Operators Classified by Relocation Attributes (Except External Reference Terms)

Table 3-11. Combination of Terms and Operators Classified by Relocation Attributes (External Reference Terms)

(8) Caution on library converter option

If a library file is converted with the library converter, it is impossible to debug object modules included in the library file.

○ Reference

Operation Volume, CHAPTER 9 LIBRARY CONVERTER

(9) Caution on assembler option

The IE-75000-R and IE-75001-R do not support source debugging. In addition, these assemblers do not distinguish uppercase and lowercase characters. Furthermore, the maximum length of a symbol name which can be recognized by these assemblers is eight characters.

○ Remedy

Specify these options as shown below.
· ~ -NGA -CA -NS

○ Reference

Operation Volume, 4.4.4 (6) -GA/-NGA, (11) -CA/-NCA, (12) -S/-NS

APPENDIX F INDEX

[MEMO]

Phase-out/Discontinued

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

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.

Thank you for your kind support.

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

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

South America
NEC do Brasil S.A.
Fax: +55-11-889-1689

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

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

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

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

Japan
NEC Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
Fax: 044-548-7900

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

Document title: ____________________

Document number: ____________________ Page number: __________

If possible, please fax the referenced page or drawing.

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