To our customers,

Old Company Name in Catalogs and Other Documents

On April 1^{st}, 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 1^{st}, 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.

RENESAS

USER'S MANUAL

RA78K SERIES ASSEMBLER PACKAGE FOR LANGUAGE

Document No. EEU-1283A
(O. D. No. EEU-654B)
Date Published October 1990P
Printed in Japan

USER'S MANUAL

NEC

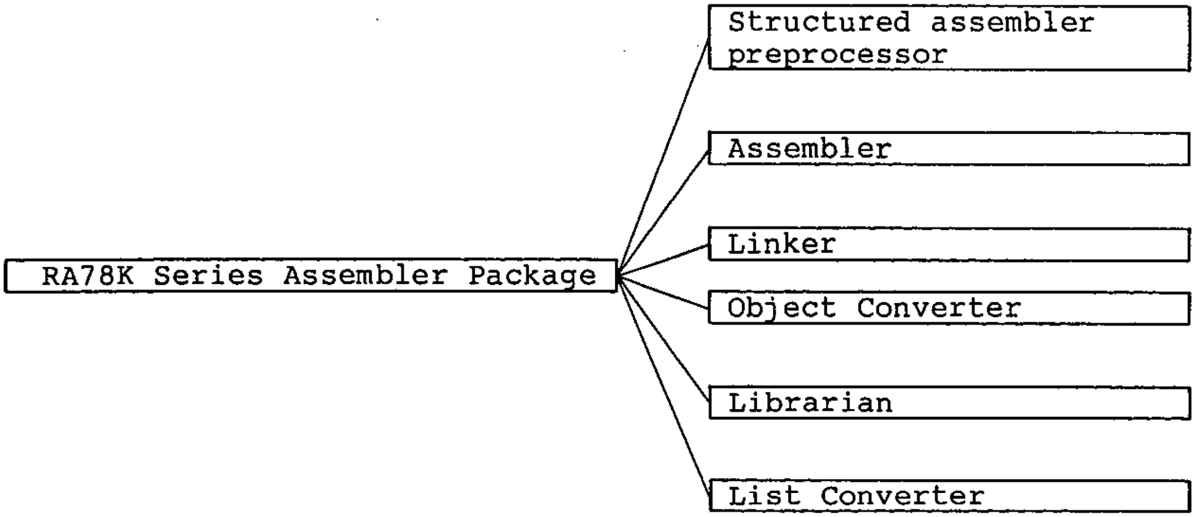
RA78K SERIES ASSEMBLER PACKAGE FOR LANGUAGE

INTRODUCTION

This manual is designed to facilitate correct understanding of the basic functions of each program in the RA78K Series Assembler Package (hereinafter referred to as "this package or the package") and the methods of describing source programs for the RA78K Series.

This manual does not cover how to operate the respective programs of the RA78K series assembler package. Therefore, after you have comprehended the contents of this manual, read the RA78K Series Assembler Package User's Manual for Operation (hereinafter referred to as "the Operation Manual") to operate each program in the assembler package. (Because the Operation Manual has been published in separate editions for the operating environments of the respective assembler packages, use the operation manual applicable to the operating environment of your assembler package.)

Descriptions relating to the RA78K/I, RA78K/II and RA78K/III in this manual are applicable to the package product versions V3.0 and upwards of the RA78K series assembler package. In all application examples in this manual, 78K/III series programs have been used.


RA78K Series Assembler Package
Structured assembler preprocessor
Assembler
Linker
Object Converter
Librarian
List Converter

[Target Devices]
The software of the following microcomputers can be developed with this package:

Package	Device	
RA78K/0	78K/0 series:	uPD78012, uPD78014
RA78K/I	78K/I series:	uPD78112 uPD78134, uPD78136, uPD78138
RA78K/II	78K/II series:	uPD78210, uPD78212, uPD78213, uPD78214 uPD78220, uPD78224 uPD78233, uPD78233
RA78K/III	78K/III series:	uPD78310A, uPD78312A uPD78320, uPD78322, uPD78330. uPD78334
RA78K/VI	78K/VI series	uPD78600, uPD78602

[Readers of Manual]
Although this manual is intended for those who are familiar with the functions and instructions of the microcomputer subject to software development, the manual can also be used by those who use an assembler program for the first time.

[Organization of Manual]
This manual consists of the following six chapters and appendixes:

Chapter 1 - General
Outlines the functions of this package including the role of the package in microcomputer development.

Chapter 2 - How to Describe Source Programs
Describes the general rules applicable to the description of a source program such as the basic configuration and description format of source programs, and the expressions and operators of the assembler.

Chapter 3 - Directives
Details the description format, function, and usage of each of the assembler directives, including application examples.

Chapter 4 - Control Instructions
Details the description format, function, and usage of each of the assembler control instructions, including application examples.

Chapter 5 - Macros
Outlines macro functions such as macrodefinition, macro reference (macrocall), and macroexpansion.
Macro directives are also explained in Chapter 3.

Chapter 6 - Product Utilization
Introduces some measures recommended for effective utilization of this package.

Appendixes
Contain a list of reserved words, a list of directives, and maximum performance characteristics.
The 78K series instruction sets are not detailed in this manual. For these instructions, refer to the user's manual of each microcomputer subject to software development.

[Recommended Usage of Manual]
For those who use an assembler for the first time: Read from Chapter 1, General of this manual.
For those who have a general understanding of assembler programs: You may skip Chapter 1, General of this manual. (However, it is advisable to read Section 1.3, "Reminders Before Program Development".)
Source programs for the 78K series can be described in several different ways. Be sure to read Chapter 2, "How to Describe Source Programs".

For those which wish to know the directives and control instructions of the assembler: Read Chapters 3 and 4, respectively, because the format, function, use, and application examples of each directive or control instruction are detailed in these chapters. A list of directives is provided in Appendix B. Use this list for quick reference.

[Symbols and Abbreviations]
The following symbols and abbreviations are used in this manual:

Symbol	Meaning
...	Continuation (repetition) of data in the same format
[ ]	Parameter(s) in brackets can be omitted.
' '	Characters enclosed in ' ' (single quotes) must be input as is.
" "	Characters enclosed in " " (double quotes) must be input as is.
( )	Characters enclosed in parentheses must be input as is.
< >	Characters enclosed in < > must be input as is (or indicates a title).
———	Important point
△	Indicates one or more Blank or TAB characters.
⋮	This part of the program description is omitted.

TABLE OF CONTENTS

APPENDIXES

LIST OF ILLUSTRATIONS AND TABLES

CHAPTER 1. GENERAL

1.1 Assembler Overview

The RA78K Series Assembler Package is a series of programs designed to translate each source program coded in the assembly language for the 78K series microprocessors, into machine language coding.

The assembler package contains six programs: Structured Assembler Preprocessor, Assembler, Linker, Object Converter, Librarian, and List Converter.

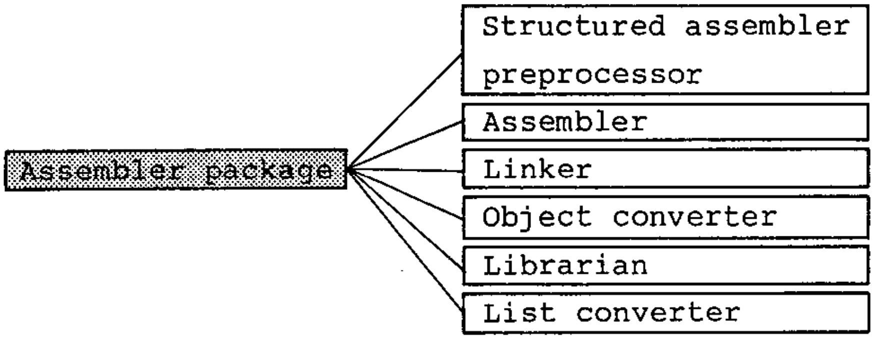


Fig. 1-1. Assembler Package

1.1.1 What is an assembler?

(1) Assembly language and machine language

An assembly language is the most fundamental programming language for microprocessors.

To have a microprocessor do its job, programs and data are required. These programs and data must be written by a human being (i.e., a programmer) and stored in the memory section of a microcomputer. Programs and data that can be handled by the microcomputer is nothing but a set or combinations of binary numbers which is called machine language (i.e., the language that can be understood or interpreted by the computer).

To create a program in machine language coding, namely, by using a set of binary numbers is not an easy job for a programmer, because it's difficult for the programmer to remember the coding and the programmer is likely to make errors in coding.

Because assembly language instructions are in one-to-one correspondence with machine language instructions, the assembly language can give the computer a detailed or specific instruction (for example, improving the I/O processing speed). For this reason, there is a method of creating a program using an abbreviated symbol (or mnemonic symbol) which represents the meaning of a machine language instruction to assist the human memory. A programming language system by this symbolic coding is called an assembly language.

To translate a program created in the assembly language into a set of binary numbers that can be understood by the microprocessor, another program is required. This program is called an assembler.

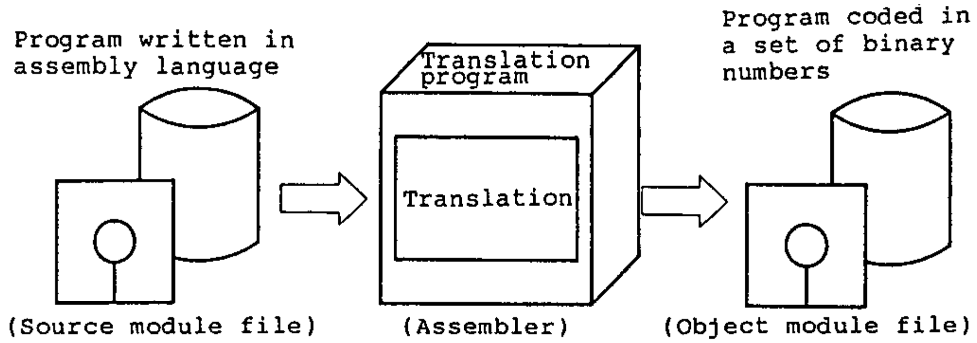


Fig. 1-2. Flow of Assembler

(2) Development of microcomputer-applied products and role of this package

Fig. 1-3 illustrates the standing of the programming in assembly language in the development process of microcomputer-applied product.

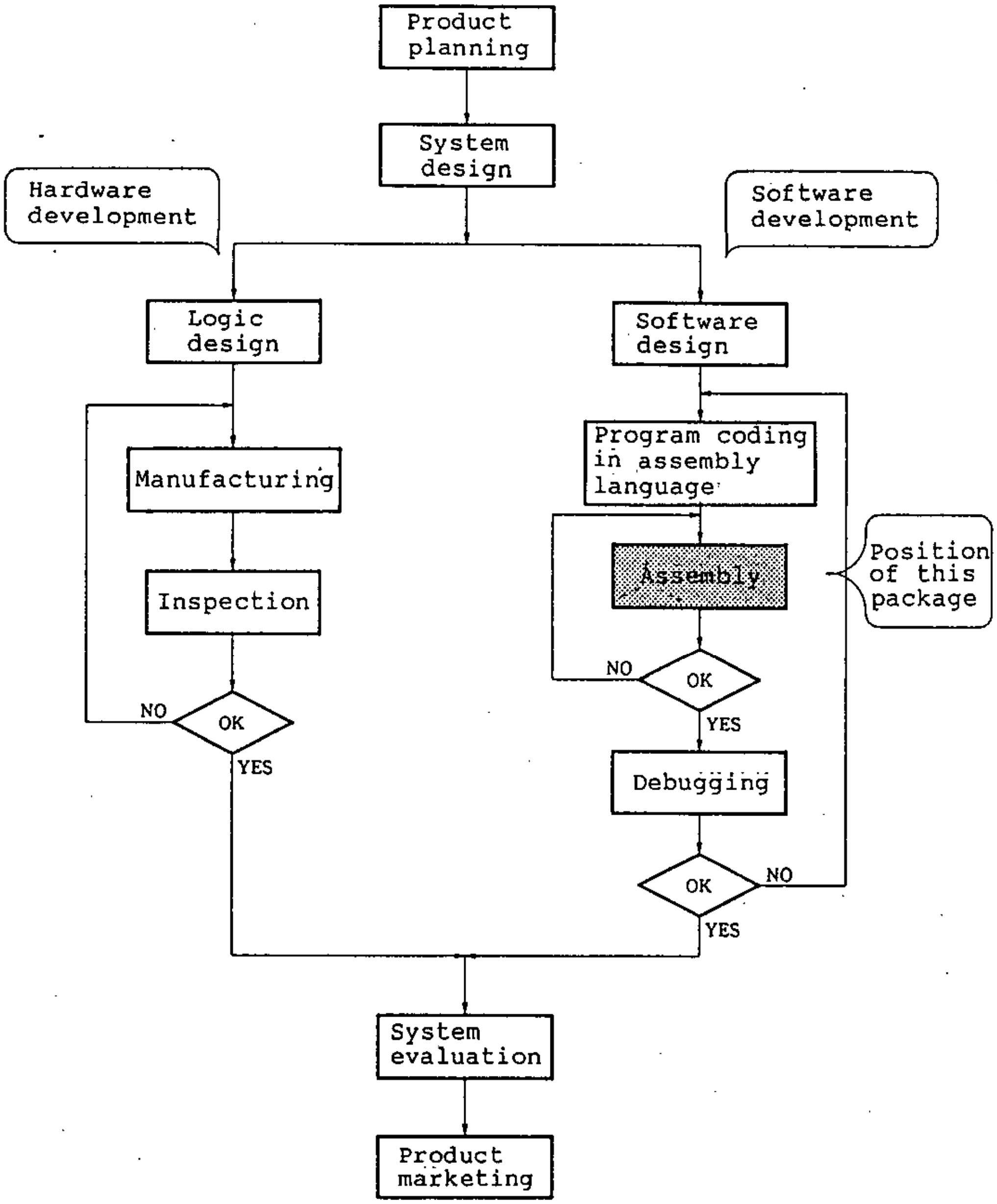
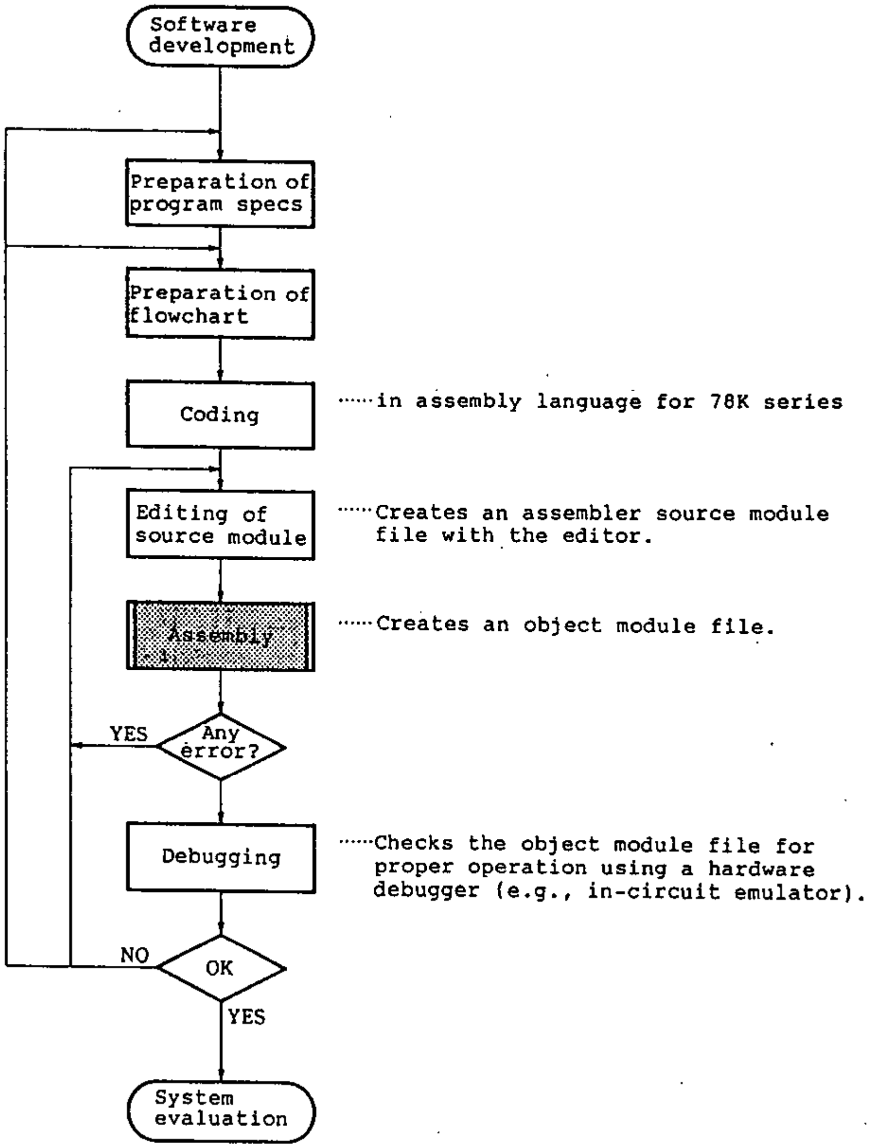


Fig. 1-3. Development Process of Microcomputer-applied Product

The software development process will be further detailed in Fig. 1-4 below.



......in assembly language for 78K series

......Creates an assembler source module file with the editor.

......Creates an object module file.

......Checks the object module file for proper operation using a hardware debugger (e.g., in-circuit emulator).

Fig. 1-4. Software Development Process

The assembly phase in the software development process will be reviewed in further detail by giving an example of this package.

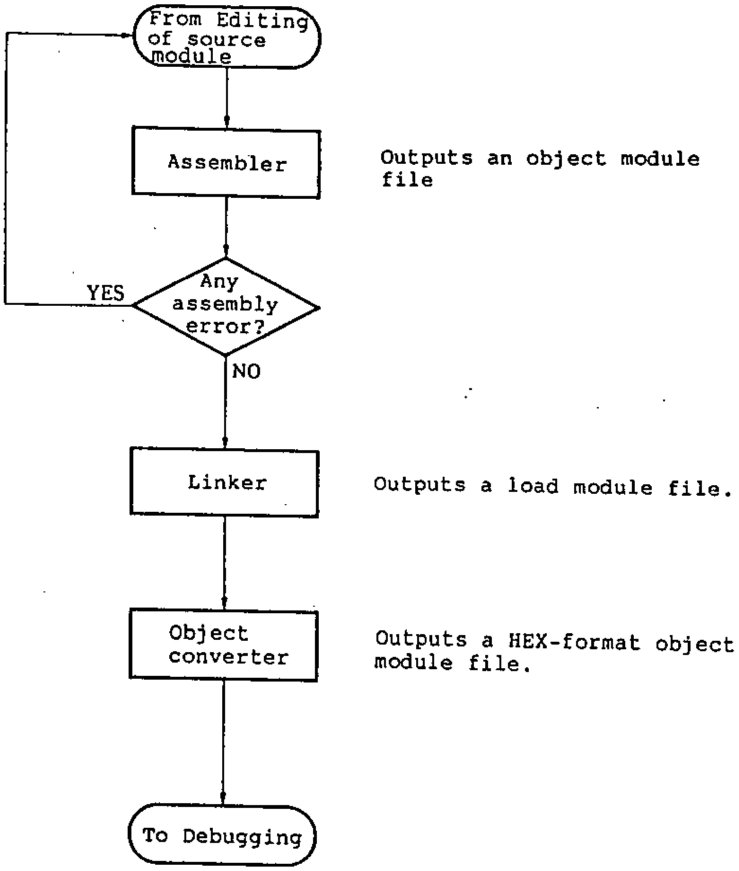

From Editing of source module
Assembler
Outputs an object module file
Any assembly error?
YES
NO
Linker
Outputs a load module file.
Object converter
Outputs a HEX-format object module file.
To Debugging

Fig. 1-5. Assembly Phase by This Package

1.1.2 What is a relocatable assembler?

The machine language translated from a source language by the assembler will be stored in the memory of the microcomputer before use. In this case, in which memory location each machine language instruction will be stored must have been determined. Therefore, information on "the allocation of each machine language instruction to a specific address in memory" will be added to the machine language converted by the assembler.

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

- o Absolute assembler
  Allocates the machine language instructions converted in one-time assembly operation to absolute addresses.
- o Relocatable assembler
  Addresses determined for the machine language instructions converted in one-time assembly operation are tentative. Absolute addresses will be determined by a program called the linker.

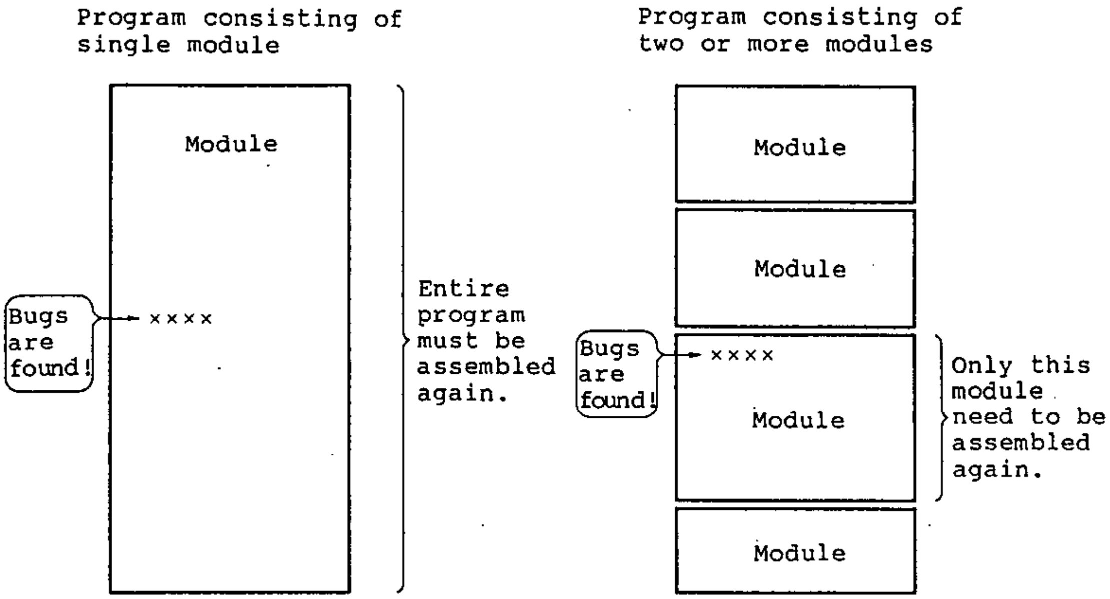
In the past, when a program was created with the absolute assembler, programmers had to, as a rule, complete programming at a time. However, if you create a large program at a time, the program becomes complicated, making analysis and maintenance of the program troublesome. To avoid this, such a large program is developed by dividing it into several subprograms (i.e., modules) for each functional unit. This programming technique is called the modular programming.

The relocatable assembler is an assembler suitable for modular programming. The following advantages can be derived from modular programming with the relocatable assembler:

(1) Increase in development efficiency

It's difficult to write a large program at a time.
In such a case, divide the program into modules for each function and the program can be developed with two or more programmers engaged in writing subprograms at the same time. This will certainly increase development efficiency of the program.

If any bugs are found in the program, you do not need to re-assemble the entire program just to correct part of the program. Only the subprogram (module) requiring correction(s) can be re-assembled. This will help shorten the debugging time.



Program consisting of single module
Module
Bugs are found!
xxxx
Entire program must be assembled again.
Program consisting of two or more modules
Module
Module
Bugs are found!
xxxx
Module
Only this module need to be assembled again.
Module

Fig. 1-6. Re-assembly for Debugging

(2) Utilization of resources

Highly reliable, highly versatile modules which have been previously created can be utilized for creation of another program. If you accumulate such high-versatility modules as software resources, you can save time and labor in developing a new program.

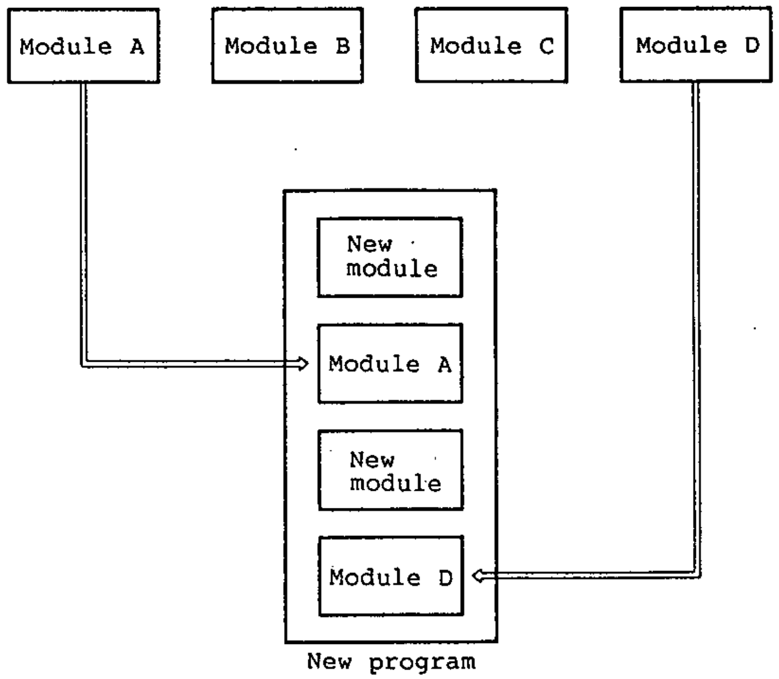


Fig. 1-7. Program Development Utilizing Existing Modules

1.2 Functional Outline of Assembler Package

An ordinary program development procedure with this assembler package is illustrated in Fig. 1-8. The development of a program is basically performed by using assembler, linker, and object converter programs.

Hereafter, programs such as Assembler, Librarian and List Converter are collectively referred to as "the assembler package or this package" and the assembly program is referred to as the assembler.

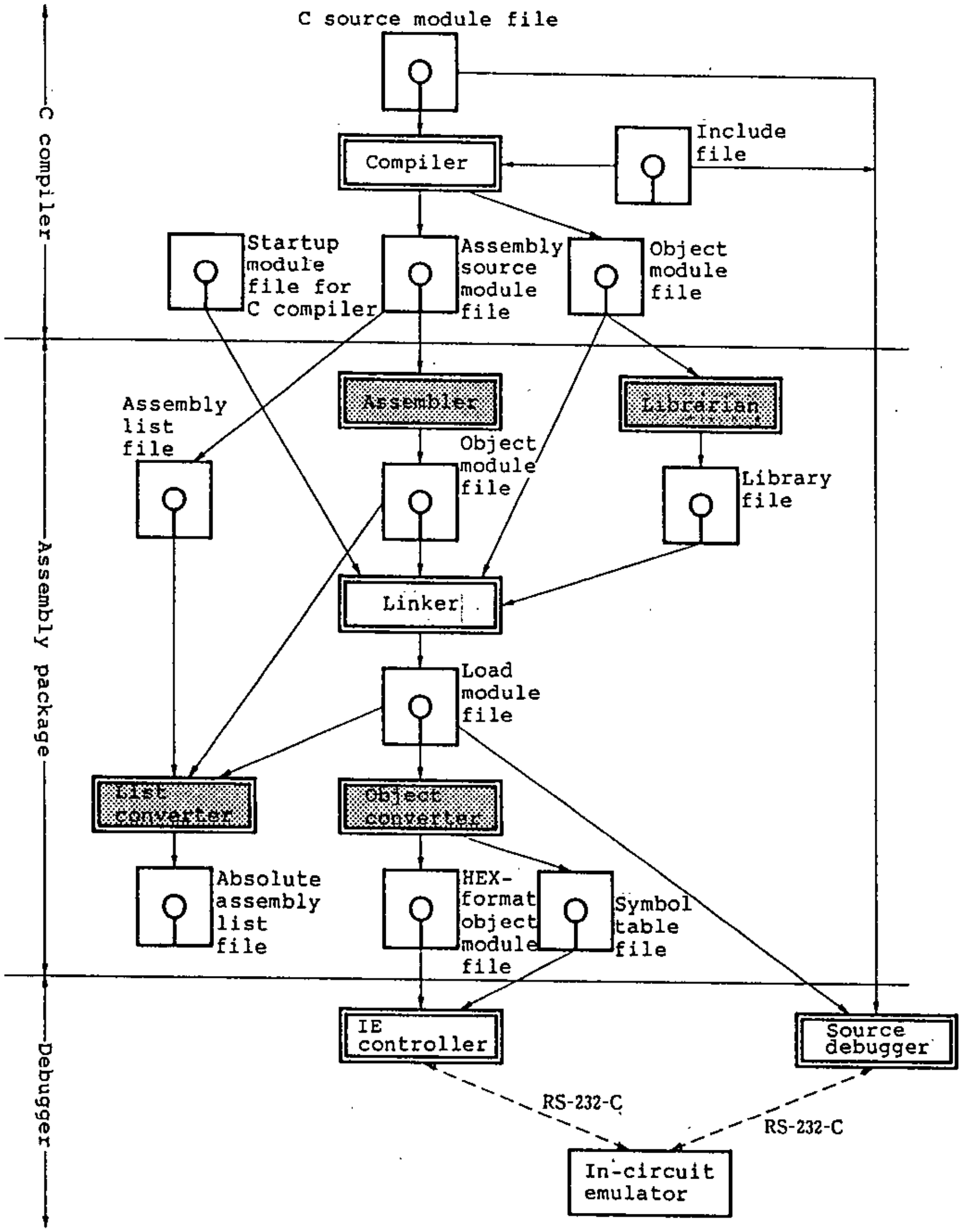


Fig. 1-8. Program Development Procedure with This Package

1.2.1 Creation of source module file with editor

Divide one program functionally into several modules.

Each module becomes the unit of coding as well as the unit of input to the assembler. A module serving as the unit of input to the assembler is called a source module.

After coding each source module, the source module is written into a file with the editor. The file thus created is called a source module file.

The source module file becomes an input file to the assembler.

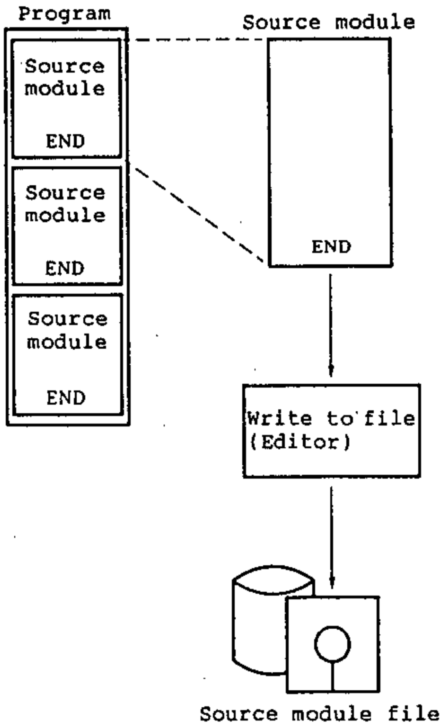


Fig. 1-9. Creation of Source Module File

1.2.2 Structured assembler preprocessor

The structured assembler preprocessor is a program for implementing structured programming in the assembly language. This program accepts a source program written in the structured assembly language as an input file and outputs an assembler source module file.

For details of the structured assembler preprocessor and structured assembly language, see the ST78K Series Structured Assembler Preprocessor User's Manual published separately.

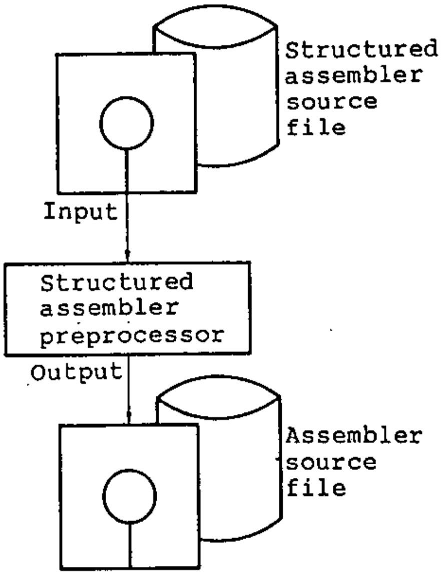

Structured assembler source file
Input
Structured assembler preprocessor
Output
Assembler source file

Fig. 1-10. Function of Structured Assembler Preprocessor

1.2.3 Assembler

The assembler accepts assembler source module files as input files and translates assembly language into machine language (a set of binary numbers). If any coding error is found in the input source module, the assembler outputs an assembly error. If no assembly error is found, the assembler outputs an object module file which contains machine language information and relocation information relating to the allocation address of each machine language instruction. The assembler also outputs information at assembly time as an assembly list file.

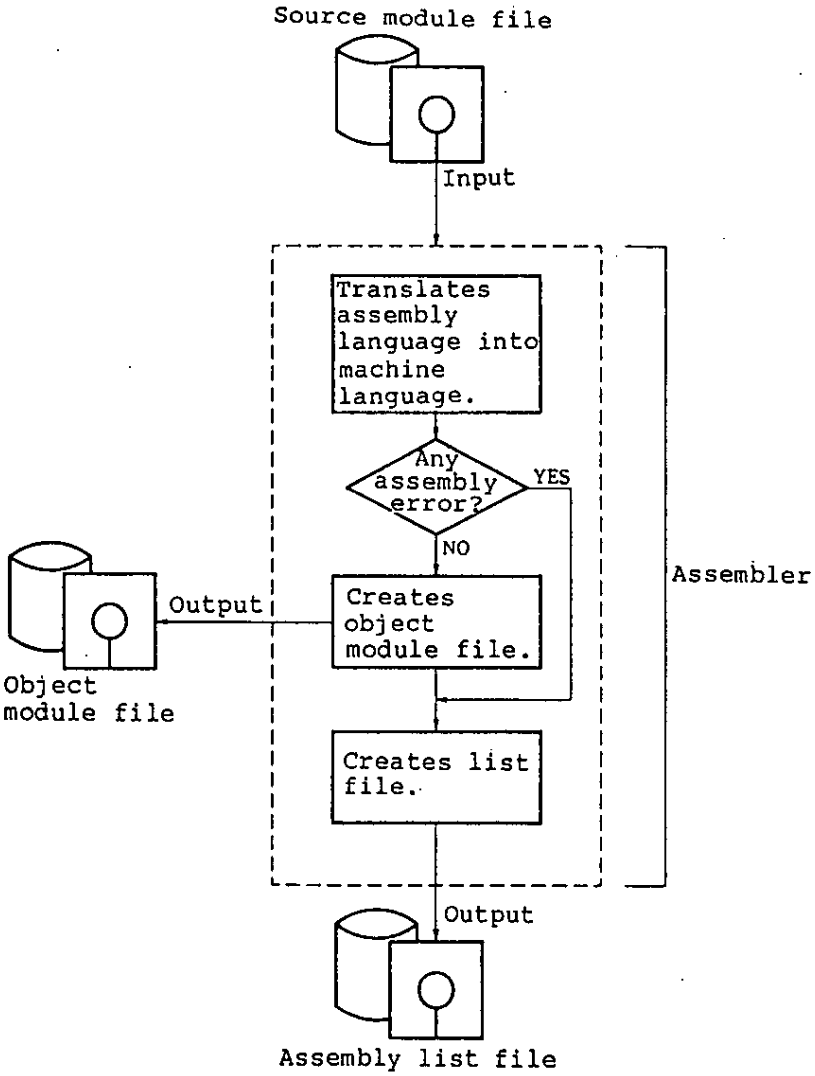


Fig. 1-11. Functions of Assembler

1.2.4 Linker

The linker accepts two or more object module files output by the compiler or assembler as input files and combines them with a library file for output as a single load module file.

The linker also determines addresses to be allocated to each relocatable segment in the input module, whereby the correct values of the respective relocatable symbols and external reference symbols are determined and embedded into the output load module file.

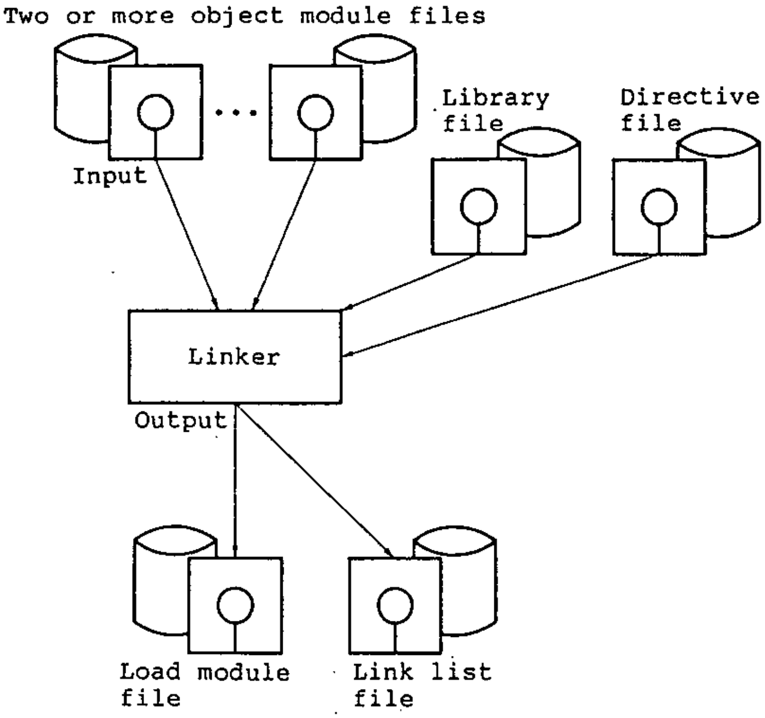


Fig. 1-12. Functions of Linker

1.2.5 Object converter

The object converter accepts the load module file output by the linker as an input file, converts its file format, and outputs the result of the conversion as an HEX-format object module file. The object converter also outputs the symbol information required in symbolic debugging as a symbol table file.

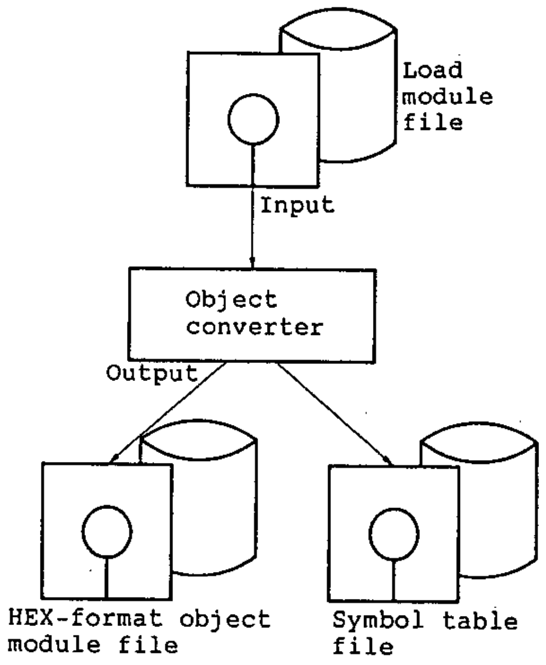


Fig. 1-13. Functions of Object Converter

1.2.6 Librarian

Modules (programs) which have versatility and a definitive interface should be kept in a single library file. By so doing, a number of object module files in a single file can be handled with ease.

The linker has a function to extract only the required modules from the library file and link them with the input object module file(s). Therefore, if you register (store) two or more modules in a single library file, you do not need to specify the required module names one by one at linking time.

The librarian is used to create and update a library file.

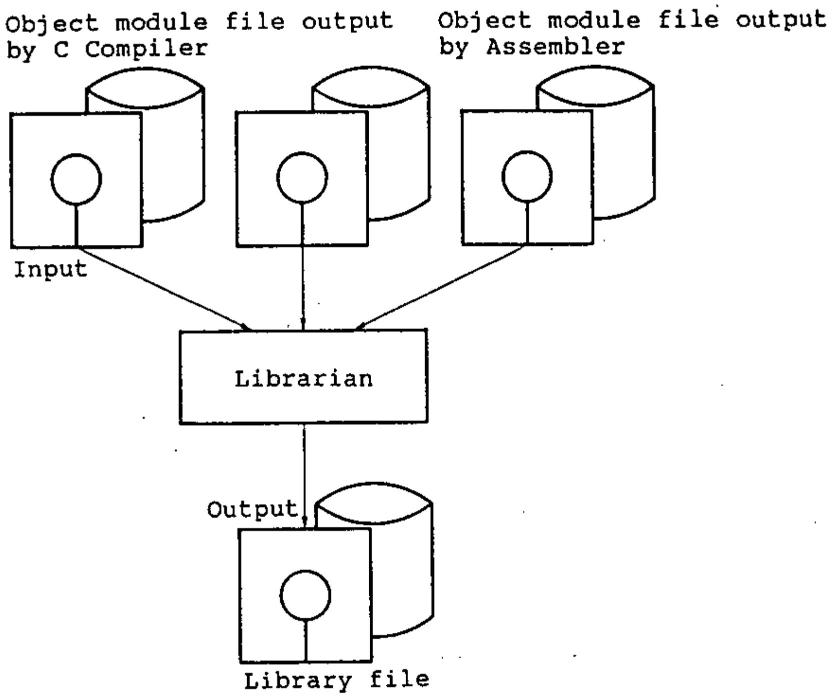

Object module file output by C Compiler
Object module file output by Assembler
Input
Librarian
Output
Library file

Fig. 1-14. Functions of Librarian

1.2.7 List converter

The list converter accepts the object module file and assembly list file output by the assembler and the load module file output by the linker as input files and outputs an absolute assembly list file.

One drawback of a relocatable assembly list is such that address values and relocatable values in the list differ from the actual values. Because an absolute assembly list has no such drawback, the absolute assembly list output by the list converter will facilitate program debugging as well as program maintenance.

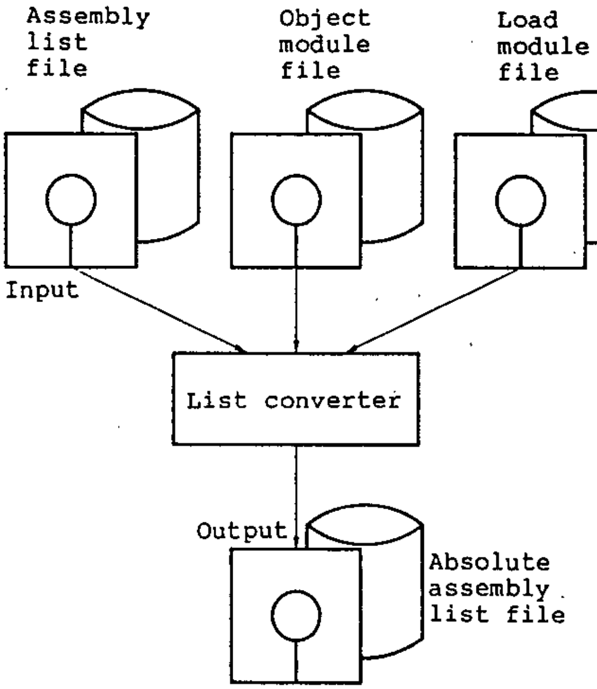


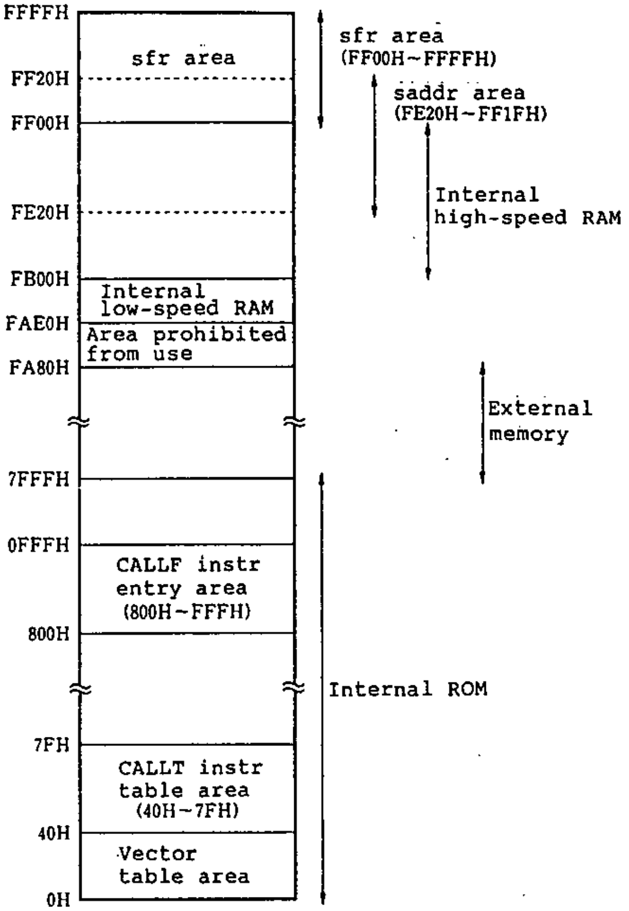
Fig. 1-15. Functions of List Converter

1.3 Memory Maps

The memory maps of the respective series in this package are shown in this section.

(1) Memory map of 78K/0

The memory map of the uPD78014 is shown below.

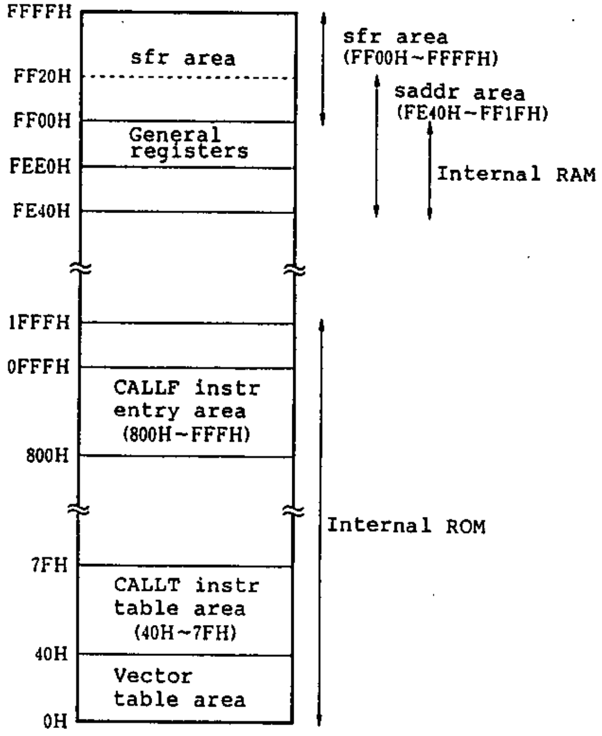


The internal ROM/RAM areas applicable to each target device in the 78K/0 series are as listed below.

Target device	Internal ROM area	Internal high-speed RAM area	Internal low-speed RAM area
uPD78012	0000H to 3FFFH	FD00H to FEFFH	FAE0H to FAFFH
uPD78014	0000H to 7FFFH	FB00H to FEFFH	FAE0H to FAFFH

(2) Memory maps of 78K/I and 78K/II

The memory map of uPD78112 is shown below.



The internal ROM/RAM areas applicable to each target device in the 78K/I series are as listed below.

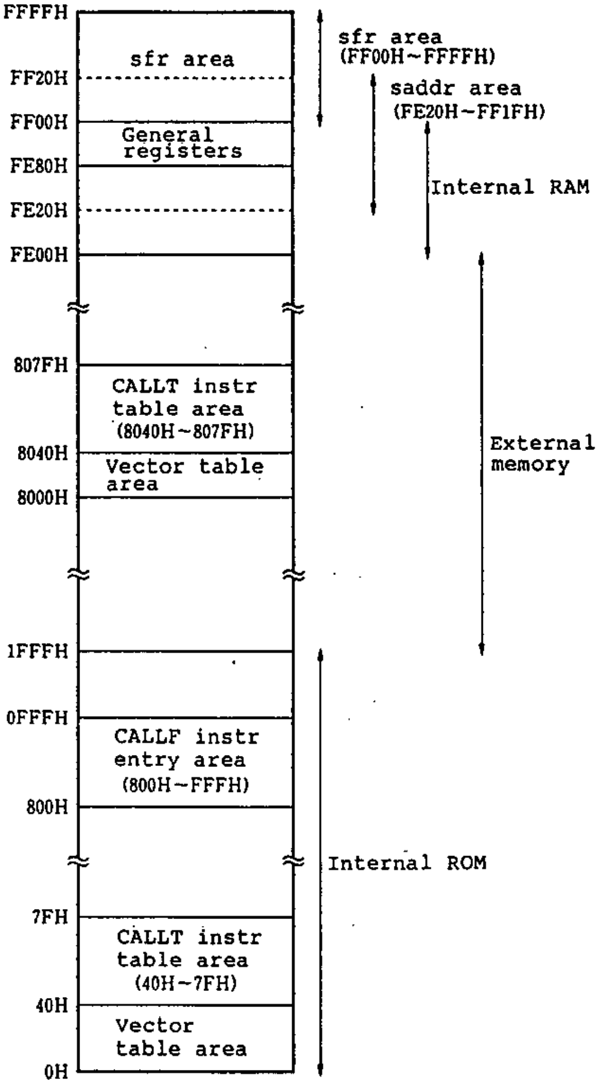
Target device	Internal ROM area	Internal RAM area	External memory	Extended memory
uPD78112	0000H to 1FFFH	FE40H to FEFFH	None	None
uPD78134/ uPD78134A	0000H to 3FFFH	FD80H to FEFFH		
uPD78136	0000H to 5FFFH	FD00H to FEFFH		
uPD78138	0000H to 7FFFH	FC80H to FEFFH		

The internal ROM/RAM areas applicable to each target device in the 78K/II series as as listed below.

Target device	Internal ROM area	Internal RAM area	External memory	Extended memory
uPD78210	None	FE80H to FEFFH	0000H to FE7FH	10000H to FFFFFH
uPD78212	0000H to 1FFFH	FD80H to FEFFH	2000H to FD7FH	
uPD78213	None	FD00H to FEFFH	0000H to FCFFH	
uPD78214	0000H to 3FFFH	FD00H to FEFFH	4000H to FCFFH	
uPD78220	None	FC80H to FEFFH	0000H to FC7FH	
uPD78224	0000H to 3FFFH	FC80H to FEFFH	4000H to FC7FH	
uPD78233	None	FC80H to FEFFH	0000H to FC7FH	
uPD78234	0000H to 3FFFH	FC80H to FEFFH	4000H to FC7FH	

(3) Memory map of 78K/III

The memory map of the uPD78312A is shown below.

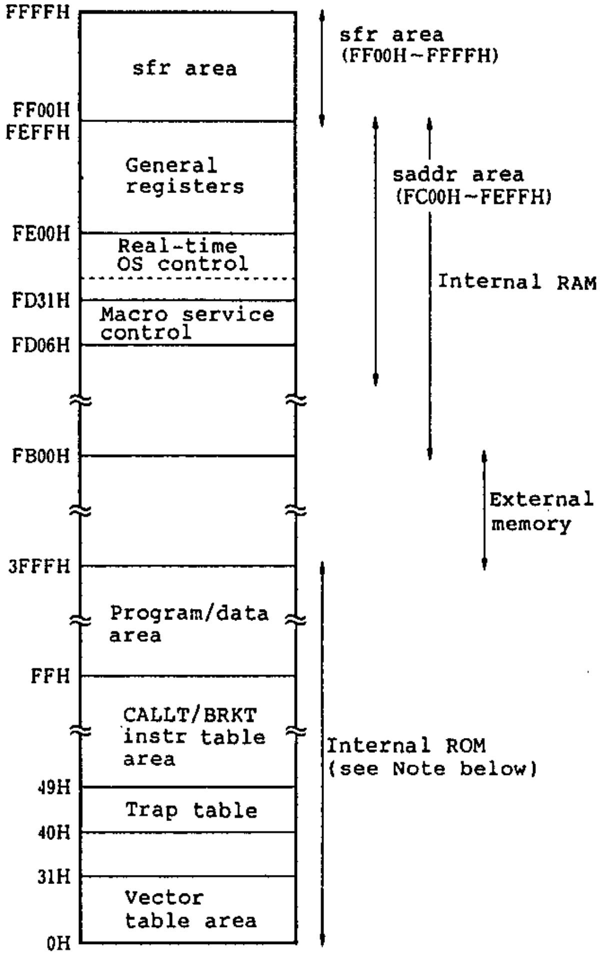


The internal ROM/RAM areas applicable to each target device in the 78K/III series are as listed below.

Target device	Internal ROM area	Internal RAM area
uPD78310/ uPD78310A	None	FE00H to FEFFH
uPD78312/ uPD78312A	0000H to 1FFFH	FE00H to FEFFH
uPD78320	None	FC80H to FEFFH
uPD78322	0000H to 3FFFH	FC80H to FEFFH
uPD78330	None	FB00H to FEFFH
uPD78334	0000H to 7FFFH	FB00H to FEFFH

(4) Memory map of 78K/VI

The memory map of the uPD78602 is shown below.



Note: When the EA (External Access) pin of the uPD78600 or uPD78602 is set at a Low level, the internal ROM area becomes an external memory area.

The internal ROM/RAM areas applicable to each target device in the 78K/VI series are as listed below.

Target device	Internal ROM area	Internal RAM area
uPD78600	None	FB00H to FEFFH
uPD78602	0000H to 3FFFH	FB00H to FEFFH

1.4 Reminders Before Program Development

Before you set your hand to the development of a program, keep in mind the following points:

1.4.1 Number of files than can be input to Linker

The number of object module files that can be input to the linker is as follows:

With 78K/0, 78K/I, 78K/VI: 128 files
With 78K/III : 64 files

1.4.2 Restriction on number of symbols

The number of local symbols and that of PUBLIC symbols in the assembler and linker, respectively, are restricted as shown in the table below.

	Number of symbols	
	No. of local symbols	No. of PUBLIC symbols
Assembler	2,900 (see Note 1)	
Linker	2,900 x No. of modules	3,000 (see Note 2)

NOTE: 1. There is no restriction on the number of symbols by symbol type. Undefined symbols will also be counted and included in the total number of symbols.

2. If the number of PUBLIC symbols exceeds 2,000, the execution speed slows down because of the additional time required to access a temporary file.

1.4.3 Maximum performance characteristics of assembler package

The maximum performance characteristics of the assembler package that should be kept in your mind before program development are listed in the tables below.

(1) Maximum performance characteristics of Assembler

Item		Restriction
Symbol length	w/o -S option	8 characters
	with -S option	31 characters
No. of characters per line		130 characters
No. of segments		100 segments

(2) Maximum performance characteristics of Linker

Item	Restriction
No. of input module files	64 files

1.5 Features of Assembler Package

This package has the following features:

(1) Macro function

When the same group of instructions must be described in a source program over and over again, a macro can be defined by giving a single macro name to the group of instructions.
By using this macro function, coding efficiency and readability of the program can be increased.

(2) Optimize function of branch instructions

The assembler package has an assembler directive to automatically select a branch instruction (i.e., BR directive).
To create a program with high memory efficiency, a 2-byte branch instruction must be described according to the branch destination range of the branch instruction. However, it is troublesome for the programmer to describe a branch instruction by paying attention to the branch destination range for each branching. If the BR directive is described, the assembler generates the appropriate branch instruction according to the branch destination range. This is called the optimize function of branch instructions.

(3) Conditional assembly function

With this function, part of a source program can be specified for assembly or non-assembly according to a predetermined condition. If a debug statement is described in a source program, whether or not the debug statement should be translated into machine language can be selected by setting a switch for conditional assembly. When the debug statement is no longer required, the source program can be assembled without major modifications to the program.

(4) Directive for general-purpose register selection

As representations for the 78K/III series general-purpose registers, absolute names (R0, R1, RP0, etc.) and function names (X, A, AX, etc.) are used. When describing a function name in a source program, a general-purpose register select directive must always be used. The RSS directive is provided to allow description of a function name in a source program.

CHAPTER 2. HOW TO DESCRIBE SOURCE PROGRAMS

2.1 Basic Configuration of Source Program

When a source program is described by dividing it into several modules, each module which becomes the unit of input to the assembler is called a source module. (If a source program consists of only one module, the source program means the same as the source module.)

Each source module which becomes the unit of input to the assembler consists mainly of the following three parts:

(1) Module header

(2) Module body

(3) Module tail

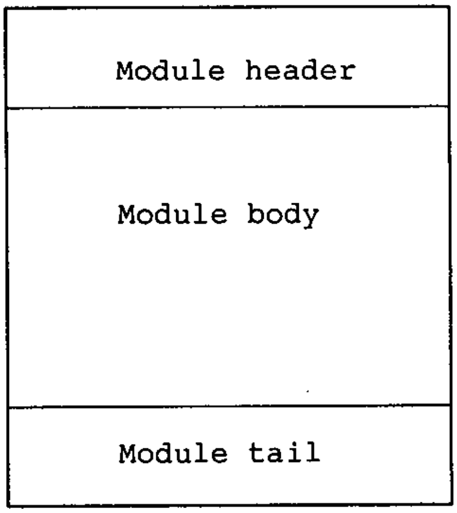


Fig. 2-1. Configuration of Source Module

2.1.1 Module header

In the module header, control instructions shown in Table 2-1 below can be described. Note that these control instructions cannot be described in other than the module header.

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

Item that can be described	Explanation	Chapter/section in this manual
Control instructions that have the same functions as assembler options	Control functions that have the same functions as assembler options include: PROCESSOR, DEBUG/NODEBUG, XREF/NOXREF, and TITLE.	See Chapter 4, "Control instructions".

2.1.2 Module body

In the module body, the following items cannot be described:

- o Control instructions that have the same functions as assembler options

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

The module body must be described by dividing it into units each called a segment.

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

(1) Code segment .... Must be defined with the CSEG directive.

(2) Data segment .... Must be defined with the DSEG directive.

(3) Bit segment .... Must be defined with the BSEG directive.

(4) Absolute segment ... Must be defined by specifying a location address for the relocation attribute (AT location address) with the CSEG, DSEG, or BSEG directive. May also be defined with the ORG directive.

The module body may be configured with any segment combinations, provided a data segment and a bit segment must be defined before a code segment.

2.1.3 Module tail

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

2.1.4 Overall configuration of source program

The overall configuration of a source module becomes as shown below.

Control instruction(s) that have the same function(s) as assembler function(s)	Module header

Directive(s)
Control instruction(s)
Instruction(s)
	Module body

END directive	Module tail

Fig. 2-2. Overall Configuration of Source Program

Examples of simple source module configurations are shown in Fig. 2-3 on the next page.

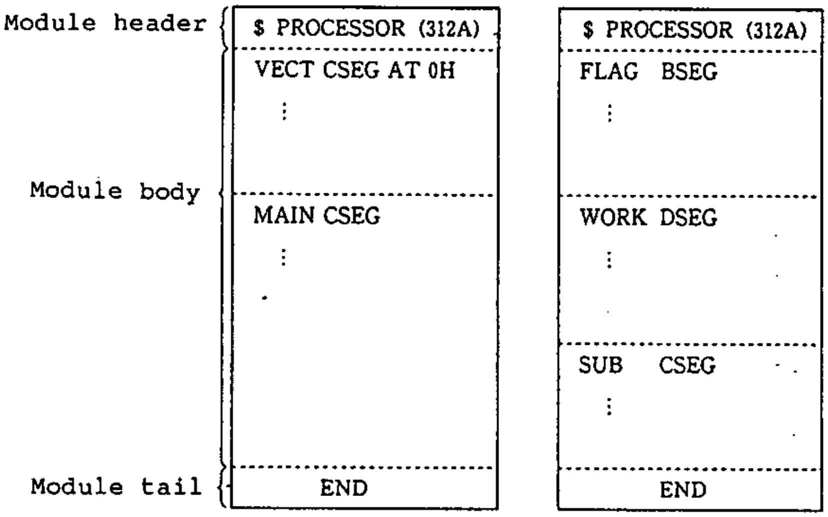


Fig. 2-3. Examples of Source Module Configurations

2.1.5 Description example of source program

In this subsection, a description example of a source program for the 78K/III series is shown. (This example is attached to the package product as a sample program file.)

The configuration of the sample program can be illustrated simply as follows:

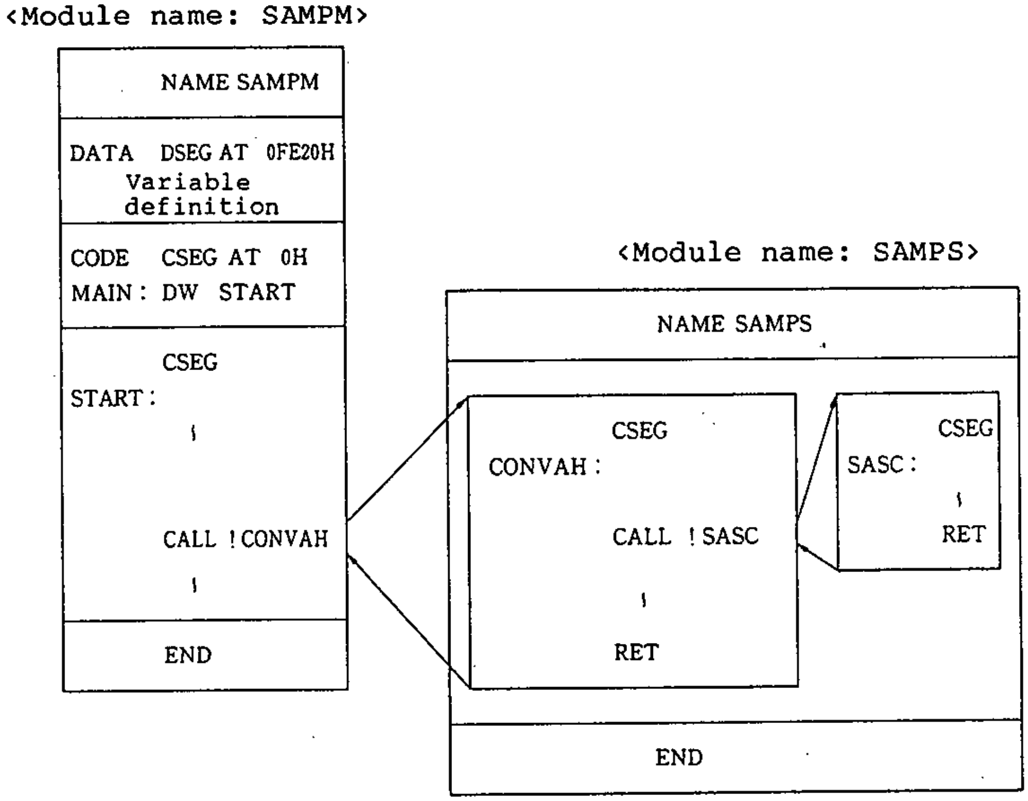
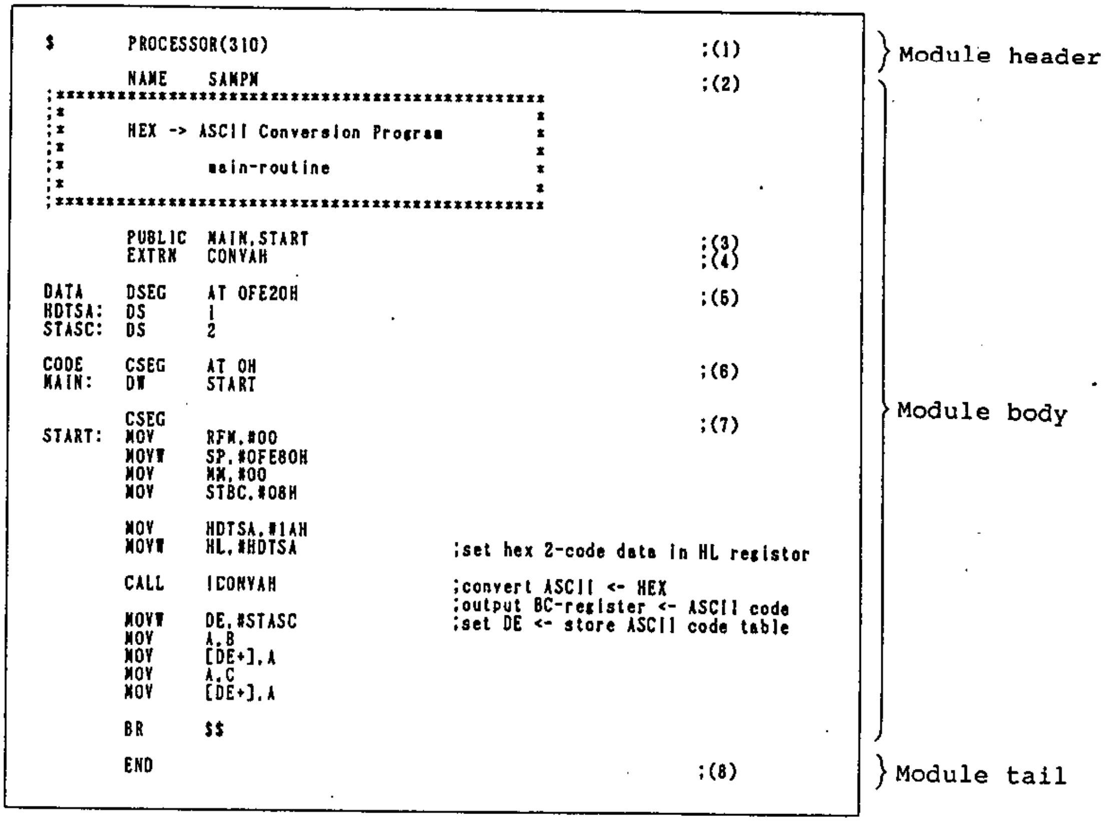


Fig. 2-4. Configuration of Sample Program

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

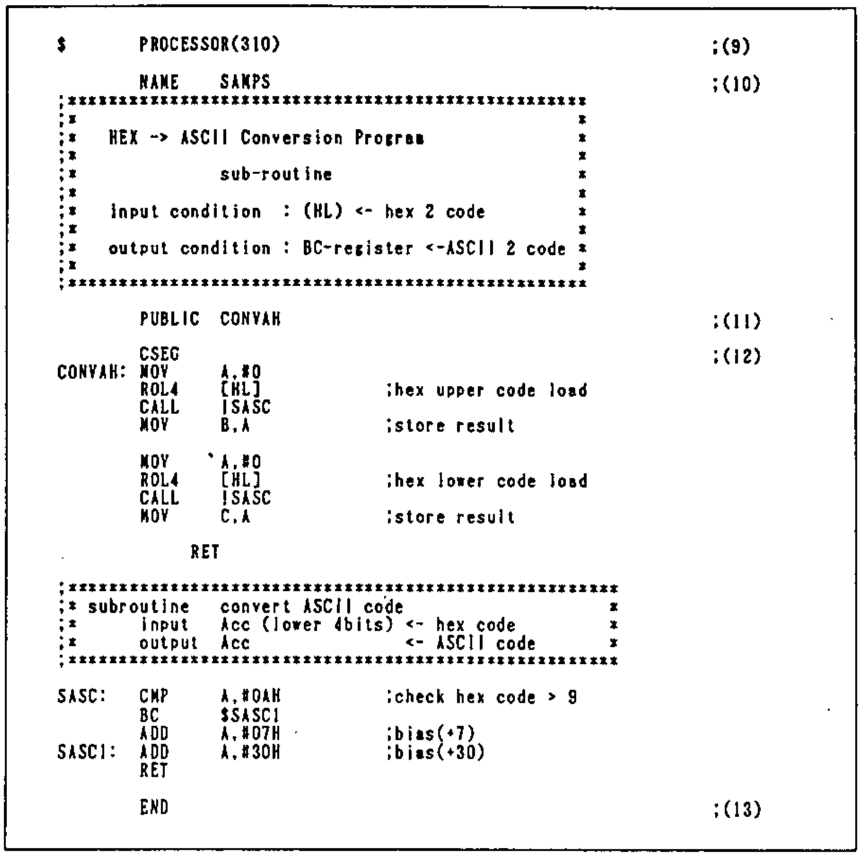
<Main routine>



```
$       PROCESSOR(310)                                    ;(1)

        NAME    SAMPN                                     ;(2)
;****************************************
;*                                      *
;*      HEX -> ASCII Conversion Program *
;*                                      *
;*              main-routine            *
;*                                      *
;****************************************

        PUBLIC  MAIN,START                                ;(3)
        EXTRN   CONVAH                                    ;(4)

DATA    DSEG    AT 0FE20H                                 ;(5)
HDTSA:  DS      1
STASC:  DS      2

CODE    CSEG    AT 0H                                     ;(6)
MAIN:   DW      START

        CSEG                                              ;(7)
START:  MOV     RFM,#00
        MOVW    SP,#0FE80H
        MOV     MM,#00
        MOV     STBC,#08H

        MOV     HDTSA,#1AH
        MOVW    HL,#HDTSA               ;set hex 2-code data in HL registor

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

        BR      $$

        END                                               ;(8)
```

(1) Control instruction which has the same function as an assembler option
(2) Declaration of a module name
(3) Declaration of a symbol referenced from another module as an external definition symbol
(4) Declaration of a symbol defined in another module as an external reference symbol
(5) Declaration of the start of a data segment (to be located as an absolute segment starting from address 0FE20H)
(6) Declaration of the start of a code segment (to be located as an absolute segment starting from address 0H)
(7) Declaration of the start of the code segment (meaning the end of the absolute segment)
(8) Declaration of the end of the module

<Subroutine>



```
$       PROCESSOR(310)                                              ;(9)

        NAME    SAMPS                                               ;(10)
;*************************************************
;*                                               *
;*   HEX -> ASCII Conversion Program             *
;*                                               *
;*             sub-routine                       *
;*                                               *
;*   input condition  : (HL) <- hex 2 code       *
;*                                               *
;*   output condition : BC-register <-ASCII 2 code *
;*                                               *
;*************************************************

        PUBLIC  CONVAH                                              ;(11)

        CSEG                                                        ;(12)
CONVAH: MOV     A,#0
        ROL4    [HL]            ;hex upper code load
        CALL    !SASC
        MOV     B,A             ;store result

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

            RET

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

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

        END                                                         ;(13)
```

} Module header

} Module body

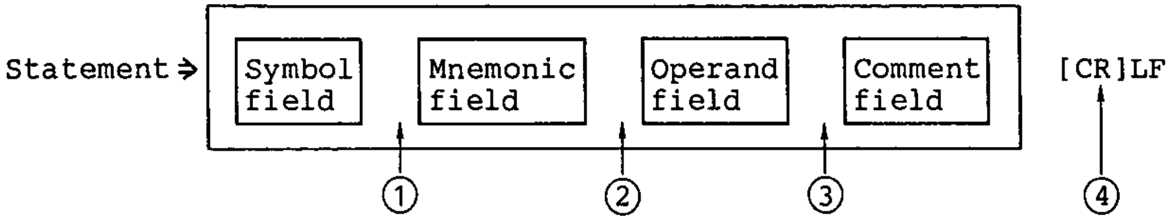
} Module tail

(9) Control instruction that has the same function as an assembler option
(10) Declaration of a module name
(11) Declaration of a symbol referenced from another module as an external definition symbol
(12) Declaration of the start of a code segment
(13) Declaration of the end of the module

2.2 Description Format of Source Program

2.2.1 Configuration of statement

A source program consists of statements.
Each statement consists of the four fields shown in Fig. 2-5.



① The Symbol field and the Mnemonic field must be separated from each other with a colon (:) or one or more blank (or TAB) characters.
② The Mnemonic field and the Operand field must be separated from each other with one or more blank (or TAB) characters. Depending on the instruction described in the Mnemonic field, the Operand field may not be required.
③ The Comment field if used must be preceded with a semicolon (;).
④ Each line must be delimited with an LF code. (One CR code may exist immediately before the LF code.)

Fig. 2-5. Fields That Make Up A Statement

A statement must be described within a line. (A line must be terminated with an LF (0AH) code.)
Up to 128 characters excluding CR and LF can be described per line. If a statement consisting of 128 or more characters is input, the assembler outputs an warning message and ignores the 129th and subsequent characters in the statement. However, in the assembly list, these ignored characters will also be output.
The following lines may also be described:

- o Dummy line (a line without statement description)
- o Line consisting of the Symbol field alone
- o Line consisting of the Comment field alone

2.2.2 Character Set

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

- o Language characters
- o Character data
- o Comment characters

(1) Language characters

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

[Alpha-numeric characters

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

NOTE 2-1

When any lowercase letter is used in a symbol or reserved word description, the lowercase letter is interpreted as its uppercase equivalent.

[Special characters]

Character	Name	Main use
?	Question mark	Symbol equivalent to alphabetic characters
@	Unit price symbol	Symbol equivalent to alphabetic characters
_	Underscore	Symbol equivalent to alphabetic characters
Blank		Delimiter of each field
HT (09H)	TAB code	Character equivalent to Blank
,	Comma	Delimiter between the 1st and 2nd operands
:	Colon	Delimiter between the Symbol and Mnemonic fields
;	Semicolon	Symbol indicating the start of the Comment field
CR (0DH)	Carriage return code	Symbol indicating the end of a line
LF (0AH)	Line-feed code	Same as above
+	Plus sign	ADD operator or positive sign
-	Minus sign	SUBTRACT operator or negative sign
*	Asterisk	MULTIPLY operator
/	Slash	DIVIDE operator
.	Period	BIT operator
( )	Left and right parentheses	Symbols specifying the order of arithmetic operations to be performed
< >	Not Equal sign	Relational operators
=	Equal sign	Relational operator
'	Single quotation mark	Symbol indicating the start or end of a character constant

Character	Name	Main use
$	Dollar sign	o Symbol indicating the location counter o Symbol indicating the start of an assembler option o Symbol specifying a relative addressing mode
#	Sharp sign	Symbol specifying an immediate addressing mode
!	Exclamation point	o Symbol specifying an absolute addressing mode o Symbol specifying the operand representation format "addr16" of an MOV instruction
[ ]	Braces	o Symbol specifying an indirect addressing mode

NOTE 2-2

If any illegal character has been described in the input source module, the assembler will replace the illegal character with "!" for output to the assembly list.

(2) Character data

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

[Character data character set]

o All characters except "00H" can be used (provided codes may be different depending on the OS). If "00H" has been described, an error will result and subsequent characters before the closing single quote (') will be ignored.

o If any illegal character has been described, the assembler will replace the illegal character with "!" for output to the assembly list. (The CR (0DH) code will not be output to the assembly list.)

- o With MS-DOS, the assembler interprets code "1AH" as the end of file (EOF) and thus the code can be a part of the input data.

(3) Comment characters

Comment characters refer to characters used to describe a comment statement.

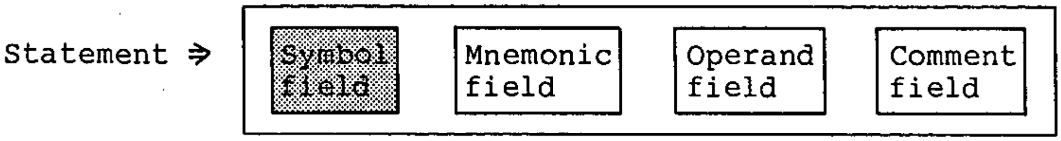
[Comment character set]

Characters in the comment character set are the same as those in the character data character set. However, no error will result even if code "00H" has been described. Instead, the assembler will output the illegal character to the assembly list by replacing it with "!".

2.2.3 Fields of Statement

The respective fields that make up a statement are detailed in this subsection.

(1) Symbol field



A symbol is described in the Symbol field. The term "symbol" refers to a name given to a numerical data or address.
By using symbols, the contents of a source program can be understood more easily.

[Symbol types]

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

Table 2-2. Symbol Types

Symbol type	Use	Method of definition
Name	Used as a numerical data in a source program.	This type is described in the Symbol field of the EQU, SET, or DBIT directive.
Label	Used as an address data in a source program.	This type is defined by suffixing a colon (:) to a symbol.
Segment name	Used as a segment name subject to operation in a linker option	This type is described in the Symbol field of the CSEG, BSEG, or ORG directive.
Module name	Used as a module name in symbolic debugging	This type is described in the Operand field of the NAME directive.
Macro name	Used as a macro name for macro reference in a source program.	This type is described in the Symbol field of the MACRO directive.

[Conventions of symbol description]

All symbols must be described according to the following rules:

① A symbol must be made up of alphanumeric characters and special characters (?, @, and _) that can be used as a symbol in a manner equivalent to alphabetic characters. As the first character of a symbol, any of the numeric characters 0 to 9 cannot be used.

② A symbol must be made up of not more than eight characters or not more than 31 characters. Which symbol length specification (1 to 8 or 1 to 31 characters) is to be used may be specified with an assembler option (-S or -NS). If a symbol is described by exceeding the maximum symbol length specified by the option, an error will result.

③ No reserved word can be used as a symbol. Reserved words are indicated in Appendix A, List of Reserved Words.

④ The same symbol cannot be described two or more times, provided that the name defined with the SET directive can be re-defined with the SET directive.

⑤ Lowercase letters described as a symbol will be interpreted by the assembler as their uppercase equivalents. However, if the assembler option -CA is specified, the assembler will distinguish between uppercase and lowercase.

⑥ When describing a label in the Symbol field, ":" (colon) must be described immediately after the label.

(Example of correct symbol descriptions)

```
TEN      EQU   10H         ; "TEN" is a name.
NEXT:    BR    !100H       ; "NEXT" is a label.
C1       CSEG              ; "C1" is a segment name.
         NAME  SAMPLE      ; "SAMPLE" is a module name.
MAC1     MACRO             ; "MAC1" is a macro name.
```

(Example of incorrect symbol descriptions)

```
   ABCDEFGHI EQU    70H          ; "I" is ignored when the maximum
                                   symbol length specification is
                                   8 characters.
   1ST:      MOV    A,#0H        ; No numeric character can be used
                                   as the 1st character of a symbol.
   NEXT      BR     !100H        ; "NEXT" is a label and must be
                                   separated from Mnemonic field
                                   with a colon (:)
   TEN       EQU    10H          ; "TEN" and "ten" are the same
   ten       EQU    20H            named symbols. Description of
                                   "ten" will thus result in an
                                   error.
```

[Symbol attributes]

Names and labels each have a value and an attribute.
Segment names, module names, and macro names have no value.
A value refers to the value of a defined numerical data or address data itself.
The attribute of a symbol is called a symbol attribute and must be one of the types and values indicated in Table 2-3.

Table 2-3. Types and Values of Symbol Attributes

Attribute type	Classification	Value
NUMBER	o Names to which numeric constants are assigned o Symbols defined with EXTRN directive	16-bit value (Cf.1) Decimal representation: -32768 to 65535 Hexadecimal representation: 0H to 0FFFFH
DNUMBER (applicable to 78K/VI only)	o Names defined with EQUD directive	32-bit value (Cf.2) Decimal representation: -214783648 to 4294967295 Hexadecimal representation: 0H to 0FFFFH
ADDRESS	o Symbols defined as labels o Names defined as labels with EQU and SET directives	16-bit value (Cf.1) Decimal representation: -32768 to 65535 Hexadecimal representation: 0H to 0FFFFH
BIT	o Names defined as bit values o Symbols defined with EXTBIT directive	78K/0, I, III: sfr or saddr area 78K/VI: 0H to 0FFFFH sfr or sfrp area Byte register Word register (Cf.3)
CSEG	Segment names defined with CSEG directive	These attribute types have no value.
DSEG	Segment names defined with DSEG directive	
BSEG	Segment names defined with BSEG directive	
MODULE	Module names defined with NAME directive (A module name if not defined is created from the primary name of the input source filename.	
MACRO	Macro names defined with MACRO directive	

Notes: 1. With an expression, each term must be a 16-bit value. An overflow in the operation on values in the expression will be ignored.

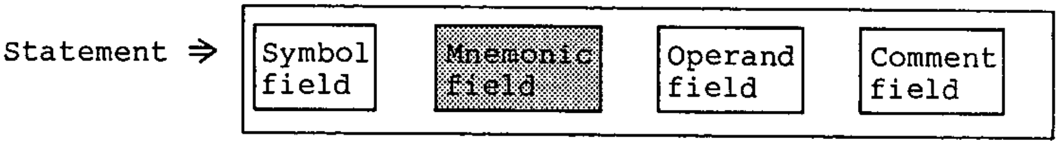
2. No expression can be described.

3. The bit position specification of the word register must be 0 to 0FH. The bit poisition specification for others must be 0 to 7.

(Examples)

```
   TEN     EQU    10H          ; Name "TEN" has attribute NUMBER
                                 and value 10H.

           ORG    80H
   START:  MOV    A,#10H       ; Label "START" has attribute
                                 ADDRESS and value 80H.

   BIT1    EQU    0FE20H. 0    ; Name "BIT1" has attribute BIT and
                                 value 0FE20H. 0.
```

(2) Mnemonic field



Statement ⇛ Symbol field | Mnemonic field | Operand field | Comment field

In the Mnemonic field, a mnemonic instruction, directive, or macro reference is described.
With an instruction or directive requiring an operand or operands, the Mnemonic field must be separated from the Operand field with one or more Blank or TAB characters.
However, with the first operand of an instruction that begins with "#", "$", "!", or "[", the assembly will be executed prperly even if nothing exists between the Mnemonic and first Operand fields.

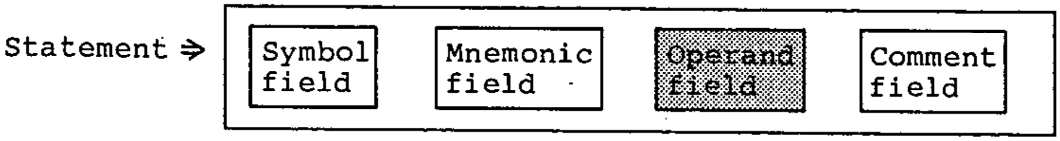
(Example of correct descriptions)

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

(Example of incorrect descriptions)

```
 MOVA,#0H          ; No blank exists between Mnemonic and
                     Operand fields.
 CAL L !CONVAH     ; A blank exists in Mnemonic field.
 HLT               ; uPD78312 has no such instruction as
                     "HLT".
```

(3) Operand field



In the Operand field, the data (operands) required for the instruction, directive, or macro reference described in the Mnemonic field must be described. Depending on the instruction or directive, no operand can be described in the Operand field or two or more operands must be described in the Operand field.

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

The following eight types of data can be described in the Operand field:

o Constants (numeric constant and string constant)
o Character strings
o Register names
o Special characters ($, #, !, and [ ])
o Relocation attributes of segment definition directives
o Symbols
o Expressions
o Bit terms

The size and attribute of the required operand may be different depending on the instruction or directive. Refer to Section 2.5, "Characteristics of Operands" for the sizes and attributes of operands.

For the operand representation formats and description methods in the 78K series instruction set, see the user's manual of the microcomputer subject to development.

Each of these eight data types that can be described in the Operand field is detailed below.

[Constants]

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

Constants are divided into numeric constants and string constants.

o Numeric constants

A binary, octal, decimal, or hexadecimal number can be described as a numeric constant. The method of representing each numeric constant type is shown in Table 2-4 below.

A numeric constant will be processed as an unsigned 16-bit data.

Value range: $0 \leqq n \leqq 65,535$ (0FFFFH)

To describe a negative value, the "-" (minus) operand must be used.

Table 2-4. Methods of Representing Numeric Constant Types

Constant type	Method of representation	Example
Binary constant	Character "B" is suffixed to a string of binary characters (value).	1101B
Octal constant	Character "O" is suffixed to a string of octal characters (value).	74O
Decimal constant	A string of decimal characters (value) may be described with or without character "D" suffixed to the string.	128 128D
Hexadecimal constant	Character "H" is suffixed to a string of hexadecimal characters (value). If the first character of the constant begins with one of the characters "A through F", "0" must be prefixed to the constant.	8CH 0A6H

- String constants

  A string constant is expressed by enclosing a character or a string of characters shown in 2.2.2, "Character set" with a pair of single quotation marks.

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

  The length of a string constant is 0 to 2.

  To use a single quotation mark as it is originally intended as a string constant, the single quotation mark must be input twice in succession.

  Examples:

```
    'A'         ; Represents "41H" (A).
    ' '         ; Represents "20H" (Space).
    ''''        ; Represents "27H" (').
    '''A'       ; Represents "2741H" ('A).
```

[Character strings]

A character string is expressed by enclosing a string of characters shown in 2.2.2, "Character set" with a pair of single quotation marks. Character strings are mainly used for operands in the DB directive and TITLE or SUBTITLE control instruction.

(Application examples of character strings)

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

[Register names]

The following registers can be described in the Operand field.

- o General-purpose registers
- o General-purpose register pairs
- o Special function registers

General-purpose registers and general-purpose register pairs can be described with their absolute names (R0 to R15 and RP1 to RP7), as well as with their function names (X, A, B, C, D, E, H, L,, AX, BC, DE, HL, VP, and UP). (However, when describing any of the general-purpose registers and general-purpose register pairs with its function name, the RSS (Register Set Select) directive must have been described. See Section 3.7, General Register Selection Directive for details of the RSS directive.)

A register name that can be described in the Operand field may be different depending on the type of instruction. See the user's manual of the microcomputer subject to development for details of the method of describing each register.

[Special characters]

Special characters that can be described in the Operand field are shown in Table 2-5.

Table 2-5. Special Characters That Can Be Described in Operand Filed

Special character	Function
$	o Indicates the location address of the instruction having this operand (or the 1st byte of the address with a multiple-byte instruction). o Indicates a relative addressing mode for a Branch instruction.
!	o Indicates an absolute addressing mode for a Branch or Call instruction. o Indicates the specification of addr16 which allows all memory space to be specified for an MOV instruction.
#	Indicates an immediate data.
[ ]	Indicates an indirect addressing mode.

(Application examples of special characters)

```
Address              Source program
100                  LOOP:   INC A
101                          BNZ $$-1     ....... 1
```

In 1 above, the first "$" in the Operand field indicates the relative addressing of the conditional branch instruction BNZ. The second "$" indicates the location address 101 to which the first byte of the object code for the instruction "BNZ $$-1" is to be assigned.

The description in 1 can be substituted with "BNZ $LOOP".

```
Source program
     BR   !100H          ; "!" indicates the absolute address-
                           ing of BR (unconditional branch)
                           instruction.
     MOV A, !2000H       ; "!" indicates addr16 specification
                           of MOV instruction
     SUB A, #10H         ; "#" indicates an immediate data.


TEN  EQU 10H
     SUB A, #TEN         ; "#" indicates an immediate data.

     AND A, [HL]         ; "[ ]" indicate an indirect
                           addressing mode.
```

[Relocation attributes of segment definition directives]

Relocation attributes may be described in the Operand field. A relocation attribute is described as the operand of a segment definition directive. By this operand, a range of location addresses for the segment can be defined.

Each segment definition directive has its own relocation attribute. For details of relocation attributes, see Section 3.2, Segment Definition Directives.

(Application examples of relocation attributes)

```
    NAME  TEST
D1  DSEG  AT   0FE20H  ; Locates data segment to address
                         FE20H.
      :
C1  CSEG  CALLT0       ; Locates code segment to addresses
                         40H to 7FH.
      :
```

[Symbols]

If a symbol is described in the Operand field, the value of the symbol becomes a numerical data subject to operation by the instruction or directive described in the Mnemonic field.

(Application examples of symbols)
```
TEN     EQU  10H
        MOV  A, #TEN      ; This description can be substituted
                            with "MOV A, #10H".
            :

WAIT    CSEG AT 100H
LOOP:   INC  A
        BNZ  $LOOP        ; This description can be substituted
                            with "BNZ $100H".
```

[Expressions]

Expressions can be described in the Operand field.
An expression is a valid series of constants, $ indicating a location address, names, or labels, that are connected with operators and can be used as an operand of an instruction.
For the expressions and operators, see Section 2.3, "Expressions and Operators".

(Application example of expression)
```
TEN   EQU   10H
      MOV   A, #TEN-5H
```

In this example, "#TEN-5H" is an expression. In this expression, name "TEN" and numeric constant "5H" are connected with the "-" (minus) operator. The value of the expression is 0BH. Therefore, this description can be substituted with "MOV A, #0BH".

[Bit terms]

Bit terms can be described in the Operand field.
A bit term may be obtained by the bit position specifier. For details of bit terms, see Section 2.4, Bit Position Specifier.

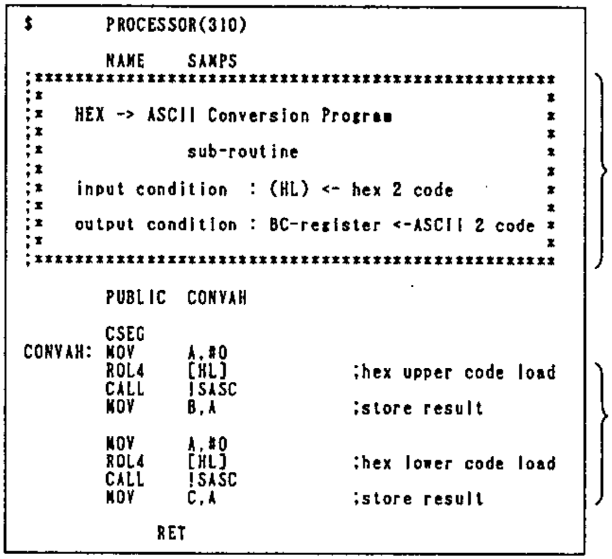
(Application examples of bit terms)
```
MOV1   CY,0FE20H.3
AND1   CY,A.5
CLR1   P1.2
SET1   1+FE30H.3       ; Equals 0FE31H.3.
SET1   0FE40H.4+2      ; Equals 0FE40H.6.
```

(4) Comment field

Statement ⇛ Symbol field | Mnemonic field | Operand field | Comment field

In the Comment field, any remarks to identify or explain a particular step or operation in a program (namely, a comment statement) may be described following the input of a semicolon (;). By describing a comment statement in the Comment field, an easy-to-understand source program can be created. The comment statement described in the Comment field is not subject to assembler operation (i.e., conversion into machine language) but will be output without change on an assembly list.
Characters that can be described in the Comment field are those shown in 2.2.2, "Character set".

(Examples of comment descriptions)



```
$       PROCESSOR(310)

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

        PUBLIC  CONVAH

        CSEG
CONVAH: MOV     A,#0
        ROL4    [HL]            ;hex upper code load
        CALL    !SASC
        MOV     B,A             ;store result

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

            RET
```

Lines consisting of Comment field only

Lines in which comments are described

2.3 Expressions and Operators

An expression is a valid series of constants, $ indicating a location address, names, or labels connected with operators. Elements of an expression other than the operators are called terms and are referred to as the 1st term, 2nd term, and so forth from left to right, in their order of description.

Operators are available in the types shown in Table 2-6, and their order of precedence in calculation has been predetermined as shown in Table 2-7.

A pair of parentheses (i.e., left and right parentheses) are used to change the order in which calculations are to be performed.

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

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

Table 2-6. Types of Operators

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

Table 2-7. Order of Precedence of Operators

Priority	Operator
Highest	
1	+ sign, - sign, NOT, HIGH, LOW
2	* (Multiply), / (Divide), MOD, SHR, SHL
3	+ (Add), - (Subtract)
4	AND
5	OR, XOR
6	EQ or =, NE or <>, GT or >, GE or >=, LT or <, LE or <=
Lowest	

Operations on expressions are performed according to the following rules:

① Operations are performed according to the order of precedence given to each operator. If two or more operators of the same order of precedence exist in an expression, the operation designated by the leftmost operator is first carried out.

② An expression in parentheses is carried out before expressions outside the parentheses.

③ Operations between two or more unary operators are allowed.

Examples:

```
1=--1=1
1=-+1=-1
```

④ Each term in an operation is handled as an unsigned 16-bit data and the result of the operation is also handled as an unsigned 16-bit.

Example:

```
65535=0FFFFH
        -1=-(0001H)=0FFFFH
```

⑤ If an overflow occurs in an operation due to its result exceeding 16 bits, only the low-order 16 bits of the result become valid. In this case, note that an error will not result.

Example:

```
65535+1=(0FFFFH)+(00001H)=(10000H)→0000H
```

If a constant exceeds 16 bits, an error will result and the value of the constant is regarded as 0 for calculation.

NOTE 2-3

No operation can be used performed on any special function register (SFR). However, as an exception in this assembler, only the LOW operator can be described for a specical fucntion name. This exception is to allow setting of the low-order 8 bits of the address of a designation or source special function register in the sfr pointer of the macro service channel. See (2) LOW in 2.3.1, Functions of Operators for how to describe the SFR name in the LOW operator.

2.3.1 Functions of Operators

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

Arithmetic Operators

(1) + (ADD) operator

Function

Returns the sum of the value of the 1st term of an expression and the value of its 2nd term.

Application Example

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

Explanation

The BR instruction causes a jump to "current address + address 6", namely, to address "100H+6H=106H".

Therefore, (a) in the above example can also be described as:

START: BR !106H

(2) - (SUBTRACT) operator

Function

Returns a difference between the value of the 1st term of an expression and the value of its 2nd term.

Application Example

```
          ORG    100H
BACK:     BR     !BACK-3H  ;(b)
           ⋮
```

Explanation

The BR instruction causes a jump to "address assigned to BACK minus 3", namely, to address "100H-3H=0FDH".

Therefore, (b) in the above example can also be described as:

BACK: BR !0FDH

(3) * (MULTIPLY) operator

Function

Returns the product of the value of the 1st term of an expression and the value of its 2nd term.

Application Example

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

Explanation

With the EQU directive, value "10H" is defined in name "TEN". "#" indicates an immediate data. Expression "TEN*3" is the same as "10H*3" and returns value 30H.

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

(4) / (DIVIDE) operator

Function

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

Application Example

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

Explanation

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

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

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

(5) MOD (Remainder) operator

Function

Obtains the remainder in the result of dividing the value of the 1st term of an expression by the value of its 2nd term. An error will result if the divisor (2nd term) is 0.
A blank is required before and after the MOD operator.

Application Example

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

Explanation

The result of division "256/50" is 5 with remainder 6.
The MOD operator returns the remainder 6.
Therefore, (e) in the above example can also be described as:
MOV A, #6

(6) + sign

Function

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

Application Example

```
FIVE   EQU   +5
```

Explanation

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

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

(7) - sign

Function

Returns the value of the term of an expression by twos complement.

Application Example

```
NO   EQU   -1
```

Explanation

-1 becomes the twos complement of 1.

```
The twos complement of binary 0000 0000 0000 0001
becomes:                      1111 1111 1111 1111
```

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

Logical Operators

(1) NOT operator

Function

Negates the value of the term of an expression on a bit-by-bit basis and returns the result.
A blank is required between the NOT operator and the term.

Application Example

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

Explanation

NOT logical operation is performed on value 3H as follows:

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

The result becomes 0FFFCH. Therefore, (a) in the above example can also be described as: MOVW AX, #0FFFCH

(2) AND operator

Function

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

Application Example

```
MOV  A, #110H  AND  0FFH  ; (b)
```

Explanation

AND operation is performed between two values 110H and 0FFH as follows:

```
        0000 0001 0001 0000
  AND)  0000 0000 1111 1111
  -------------------------
        0000 0000 0001 0000
```

The result becomes 10H. Therefore, (b) in the above example can also be described as: MOV A, #10H

(3) OR operator

Function

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

Application Example

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

Explanation

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

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

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

(4) XOR operator

Function

Performs an Exclusive-OR operation between the value of the 1st term of an expression and the value of its 2nd term on a bit-by-bit basis and returns the result.
A blank is required before and after the XOR operator.

Application Example

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

Explanation

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

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

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

Relational Operators

(1) EQ or = (Equal) operator

Function

Returns 0FFH if the value of the 1st term of an expression is equal to the value of its 2nd term (i.e., true) and 00H if both values are not equal (i.e., false).
A blank is required before and after the EQ operator.

Application Example

```
A1    EQU    12C4H
A2    EQU    12C0H

      MOV    A, #A1  EQ   (A2+4)   ;(a)
      MOV    X, #A1  EQ    A2      ;(b)
```

Explanation

In (a) above,
expression "A1 EQ (A2+4)" becomes "12C4H EQ (12C0H+4)".
The operator compares the value of the 1st term and that of the 2nd term and returns 0FFH because the value of the 1st term is equal to the value of the 2nd term.
In (b) above,
expression "A1 EQ A2" becomes "12C4H EQ 12C0H".
The operator compares the value of the 1st term and that of the 2nd term and returns 00H because the value of the 1st term is not equal to the value of the 2nd term.

(2) NE or <> (Not Equal) operator

Function

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

A blank is required before and after the NE operator.

Application Example

```
A1    EQU    5678H
A2    EQU    5670H

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

Explanation

In (c) above,

expression "A1 NE A2" becomes "5678H NE 5670H".

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

In (d) above,

expression "A1 NE (A2+8H)" becomes "5678H NE (5670H+8H)".

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

(3) GT or > (Greater-Than) operator

Function

Returns 0FFH if the value of the 1st term of an expression is greater than the value of its 2nd term (i.e., true) and 00H if the value of the 1st term is equal to or less than the value of the 2nd term (i.e., false).
A blank is required before and after the GT operator.

Application Example

```
A1    EQU    1023H
A2    EQU    1013H

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

Explanation

In (e) above,
expression "A1 GT A2" becomes "1023H GT 1013H".
The operator compares the value of the 1st term and that of the 2nd term and returns 0FFH because the value of the 1st term is greater than the value of the 2nd term.
In (f) above,
expression "A1 GT (A2+10H)" becomes "1023H GT (1013H+10H)".
The operator compares the value of the 1st term and that of the 2nd term and returns 00H because the value of the 1st term is equal to the value of the 2nd term.

(4) GE or >= (Greater-than or Equal) operator

Function

Returns 0FFH if the value of the 1st term of an expression is greater than or equal to the value of its 2nd term (i.e., true) and 00H if the value of the 1st term is less than the value of the 2nd term (i.e., false).
A blank is required before and after the GE operator.

Application Example

```
A1    EQU   2037H
A2    EQU   2015H

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

Explanation

In (g) above,
expression "A1 GE A2" becomes "2037H GE 2015H".
The operator compares the value of the 1st term and that of the 2nd term and returns 0FFH because the value of the 1st term is greater than the value of the 2nd term.
In (h) above,
expression "A1 GE (A2+23H)" becomes "2037H GE (2015H+23H)".
The operator compares the value of the 1st term and that of the 2nd term and returns 00H because the value of the 1st term is less than the value of the 2nd term.

(5) LT or < (Less-Than) operator

Function

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

A blank is required before and after the LT operator.

Application Example

```
A1      EQU     1000H
A2      EQU     1020H

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

Explanation

In (i) above,

expression "A1 LT A2" becomes "1000H LT 1020H".

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

In (j) above,

expression "(A1+20H) LT A2" becomes "(1000H+20H) LT 1020H".

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

(6) LE or <= (Less-than or Equal) operator

Function

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

A blank is required before and after the LE operator.

Application Example

```
A1   EQU   103AH
A2   EQU   1040H

     MOV   A,#A1 LE A2           ;(k)
     MOV   X,#(A1+7H) LE A2      ;(l)
```

Explanation

In (k) above,

expression "A1 LE A2" becomes "103AH LE 1040H".

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

In (l) above,

expression "(A1+7H) LE A2" becomes "(103AH+7H) LE 1040H".

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

(1) SHR (Shift Right) operator

Function

Returns a value obtained by shifting the value of the 1st term of an expression to the right the number of bits specified by the value of the 2nd term. In this case, zeroes equivalent to the specified number of bits shifted move in from the MSB side.

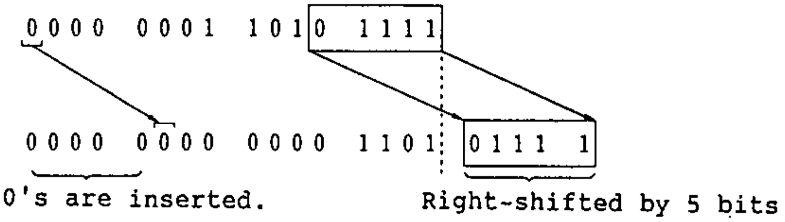
A blank is required before and after the SHR operator.

Application Example

```
MOV  A, #1AFH  SHR  5    ;(a)
```

Explanation

This operator shifts value "01AFH" to the right by 5 bits.



As the result of the Shift Right operation, value 0DH is returned. Therefore, (a) in the above example can also be described as: MOV A, #0DH

(2) SHL (Shift Left) operator

Function

Returns a value obtained by shifting the value of the 1st term of an expression to the left the number of bits specified by the value of the 2nd term. In this case, zeroes equivalent to the specified number of bits shifted move in from the LSB side.

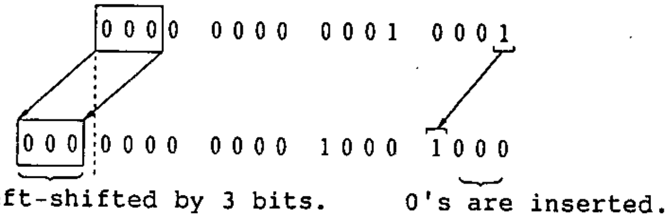
A blank is required before and after the SHL operator.

Application Example

```
MOV  A, #11H  SHL  3    ;(b)
```

Explanation

This operator shifts value "11H" to the left by 3 bits.



As the result of the Shift Left operation, value 88H is returned. Therefore, (b) in the above example can also be described as: MOV A, #88H

Byte Separating Operators

(1) HIGH operator

Function

Returns the high-order 8-bit value of the term of an expression.

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

Application Example

```
        ORG  1234H
START:
           :
        MOV  A, #HIGH START    ;(a)
```

Explanation

Because label "START" has value 1234H, the HIGH operator returns the high-order 8 bits of the value, namely, 12H. Therefore, (a) in the above example can also be described as:

MOV A, #12H

(2) LOW operator

Function

Returns the low-order 8-bit value of the term of an expression.

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

Application Example

```
        ORG  5678H
WORK:
           :
        MOV  A. #LOW WORK    ;(b)
```

Explanation

Because label "WORK" has value 5678H, the LOW operator returns the low-order 8 bits of the value, namely, 78H. Therefore, (b) in the above example can also be described as:

MOV A, #78H

Note

No operation can be used performed on any special function register (SFR). However, as an exception in this assembler, only the LOW operator can be described for a specical function name. This exception is to allow setting of the low-order 8 bits of the address of a designation or source special function register in the sfr pointer of the macro service channel.

An SFR name can be described in the LOW operator in either of the following two ways:

① LOW△SFR name

② LOW [△] ([△]SFR name[△])

The result of the LOW operation becomes an absolute term with the NUMBER attribute.

Other Operators

(1) ( )

Function

Causes an operation in parentheses to be performed prior to operations outside the parentheses.
This operator is used to change the order of precedence of other operators.
If parentheses are nested at multiple levels, the expression in the innermost parentheses will be first calculated.

Application Example

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

Explanation

$$\underbrace{\underbrace{(4+3)}_{\text{①}} * 2}_{\text{②}}$$

Calculations are performed in the order of expressions ① and ② and value 14 is returned as the result.
If parentheses are not used as shown below,

$$\underbrace{4+\underbrace{3 * 2}_{\text{①}}}_{\text{②}}$$

calculations are performed in the order of expressions ① and ② and a different value 10 is returned as the result.
See Table 2-7 for the order of precedence of operators.

2.3.2 Restrictions on Operations

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

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

(1) Operators and relocation attributes

As previously mentioned, each of the terms which constitute an expression has a relocation attribute. Terms can be divided into three types when classified by their relocation attributes: Absolute term, Relocatable term, and External reference term.

Types of relocation attributes in operations, nature of each attribute, and terms applicable to each attribute are shown in Table 2-8.

Table 2-8. Types of Relocation Attributes

Type	Nature	Applicable term
Absolute term	Term whose value is determined at assembly time	o Constants o Labels defined within an absolute segment o $ indicating the location address defined within an absolute segment o Names defined the above constants, labels, or $
Relocatable term	Term whose value is not determined at assembly time	o Labels defined within a relocatable segment o $ indicating the location address defined within a relocatable segment o Names defined relocatable labels

Table 2-8. Types of Relocation Attributes (contd)

Type	Nature	Applicable term
External reference term	Term which externally references the symbol of another module	o Labels defined with EXTRN directive o Names defined with EXTBIT directive

Combinations of the type of operator and terms on which each operator can work are shown in Table 2-9.

Table 2-9. Combinations of Operators and Terms by Relocation Attribute

Relocation attribute of term Type of operator	X: ABS Y: ABS	X: ABS Y: REL	X: REL Y: ABS	X: REL Y: REL
X + Y	A	R	R	-
X - Y	A	-	R	A*
X * Y	A	-	-	-
X / Y	A	-	-	-
X MOD Y	A	-	-	-
X SHL Y	A	-	-	-
X SHR Y	A	-	-	-
X EQ Y	A	-	-	A*
X LT Y	A	-	-	A*
X LE Y	A	-	-	A*
X GT Y	A	-	-	A*
X GE Y	A	-	-	A*
X NE Y	A	-	-	A*
X AND Y	A	-	-	-
X OR Y	A	-	-	-
X XOR Y	A	-	-	-
NOT X	A	A	-	-
+ X	A	A	R	R
- X	A	A	-	-
HIGH X	A	A	R**	R**
LOW X	A	A	R**	R**

<Legend> ABS: Absolute term
REL: Relocatable term
A : The result of the operation becomes an absolute term.
R : The result of the operation becomes an relocatable term.
- : The operation cannot be performed.

* The operation is allowed only between the symbols defined within the same segment. However, the relocatable term on which a HIGH or LOW operation is performmed will not bee regarded as the same segment.

** No term for which the HIGH or LOW operation is specified cannot be used.

The following four operators can work on external reference terms: +, -, HIGH, and LOW. (However, note that only one external reference term can be described in an expression.) Combinations of the type of operator and external reference terms on which each operator can work are shown in Table 2-10.

Table 2-10. Combinations of Operators and Terms by Relocation Attribute (External Reference Term)

Relocation attribute of term Type of operator	X: ABS Y: EXT	X: EXT Y: ABS	X: REL Y: EXT	X: EXT Y: REL	X: EXT Y: EXT
X + Y	E	E	-	-	-
X - Y	-	E	-	-	-
+ X	A	E	R	E	E
HIGH X	A	E*	R*	E*	E*
LOW X	A	E*	R*	E*	E*

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

* No term for which the HIGH or LOW operation is specified cannot be used.

(2) Operators and symbol attributes

As previously mentioned, each of the terms which constitute an expression has a symbol attribute in addition to a relocation attribute. Terms can be divided into two types when classified by their system attributes: NUMBER term and ADDRESS term. Types of system attributes in operations and terms applicable to each attribute are shown in Table 2-11.

Table 2-11. Types of Symbol Attributes

Type	Applicable term
NUMBER term	o Names and labels which have NUMBER attribute o Constants
ADDRESS term	o Names and labels which have ADDRESS attribute o $ indicating the location counter

Combinations of the type of operator and terms on which each operator can work are shown in Table 2-12.

Table 2-12. Combinations of Operators and Terms by Symbol Attribute

Symbol attribute of term	X:ADDRESS	X:ADDRESS	X:NUMBER	X:NUMBER
Type of operator	Y:ADDRESS	Y:NUMBER	Y:ADDRESS	Y:NUMBER
X + Y	-	A	A	N
X - Y	N	A	-	N
X * Y	-	-	-	N
X / Y	-	-	-	N
X MOD Y	-	-	-	N
X SHL Y	-	-	-	N
X SHR Y	-	-	-	N
X EQ Y	N	-	-	N
X LT Y	N	-	-	N
X LE Y	N	-	-	N
X GT Y	N	-	-	N
X GE Y	N	-	-	N
X NE Y	N	-	-	N
X AND Y	-	-	-	N
X OR Y	-	-	-	N
X XOR Y	-	-	-	N
NOT X	-	-	N	N
+ X	A	A	N	N
- X	-	-	N	N
HIGH X	A	A	N	N
LOW X	A	A	N	N

<Legend> ADDRESS: ADDRESS term
NUMBER : NUMBER term
A : The result of the operation becomes an ADDRESS term.
N : The result of the operation becomes a NUMBER term.
- : The operation cannot be performed.

(3) How to check restrictions on the operation

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

Example: BR $TABLE+5H

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

① Operator and relocation attribute

Because "TABLE+5H" is "relocatable term + absolute term", apply this operation to Table 2-9, "Combinations of Operators and Terms by Relocatable Attribute".

Type of operator ⟶ X + Y

Relocation attribute of term ⟶ X:REL, Y:ABS

From the table, you will find that the result becomes R (namely, a relocatable term).

② Operator and symbol attribute

Because "TABLE+5H" is "ADDRESS term + NUMBER term", apply this operation to Table 2-12, "Combinations of Operators and Terms by Symbol Attribute".

Type of operator ⟶ X + Y

Symbol attribute of term ⟶ X:ADDRESS, Y:NUMBER

From the table, you will find that the result becomes A (namely, an ADDRESS term).

2.4 Bit Position Specifier

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

(1) Period (.) (Bit position specifier)

Description Format

$$\underbrace{\texttt{X [}\triangle\texttt{], [}\triangle\texttt{] Y}}_{\text{Bit term}}$$

Combinations of X (1st term) and Y (2nd term) are shown in the following table.

X (1st term)	Y (2nd term)	78K/0	78K/I,II	78K/III	78K/VI
A register X register	expression (0 to 7)	o	o o	o o	
br* register	expression (0 to 7) or br*				o
wr* register	expression (0 to 0FH) or br*				o
PSW	expression (0 to 7)	o	o		
PSWL PSWH				o	
sfr*	expression (0 to 7)	o	o	o	
bsfr*	expression (0 to 7) or br*				o
saddr*	expression (0 to 7)	o	o	o	
bsaddr*	expression (0 to 7) or br*				o
mem*					

* For details on the specifier description, see the user's manual of each device.

Function

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

Explanation

- o A bit term refers to an expression or register specified on both sides of the period (.).
- o The bit position specifier is not affected by the precedence order of operators. The left side (expression or register) of the bit position specifier is recognized as the 1st term and its right side (expression) as the 2nd term.
- o There are the following restrictions on the 1st term:
  - ① An expression with the NUMBER or ADDRESS attribute can be described on the left side of the period.
  - ② An externally referenced symbol can also be described on the left side of the period.
- o There are the following restrictions on the 2nd term:
  - ① The value of an expression described on the right side of the period must be in the range of 0 to 7. If this value range is exceeded, an error will result.
  - ② Only an absolute expression with the NUMBER attribute can be described on the right side of the period.
  - ③ No externally referenced symbol can be used on the right side of the period.

Operations and Relocation Attributes

Combinations of the 1st and 2nd terms by relocation attribute are shown in Table 2-13.

Table 2-13. Combinations of 1st and 2nd Terms by Relocation Attribute

Combination of X:	ABS	ABS	REL	REL	ABS	EXT	REL	EXT	EXT
terms Y:	ABS	REL	ABS	REL	EXT	ABS	EXT	REL	EXT
X. Y	A	-	R	-	-	E	-	-	-

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

Values of Bit Symbols

If a bit symbol is defined by describing a bit term using the bit position specifier in the operand field of the EQU directive, the value that the bit symbol will have is shown in Table 2-14 below.

Table 2-14. Values of Bit Symbols

Operand type	Symbol value	78K/0	78K/I,II	78K/III	78K/VI
A.bit1 (Cf.2)	1.bit1	o	o	o	
X.bit1 (Cf.2)	0.bit1		o	o	
br.bit1 (Cf.1 & 2)	register-name-value1.bit1 (Cf.3)				o
wr.bit2 (Cf.1 & 2)	register-name-value2.bit2 (Cf.3)				
PSW.bit1 (Cf.2)	1FEH.bit1	o	o		
PSWL.bit1(Cf.2)				o	
PSWH.bit1(Cf.2)	1FFH.bit1				
sfr.bit1 (Cf.1 & 2)	0FFxxH.bit1 (Cf.4)	o	o	o	
bsfr.bit1 (Cf.1 & 2)					o
expression. bit1 (Cf.2)	0xxxxH. bit1 (Cf.4)				o

Notes: 1. For details of the bit symbol description, see the user's manual of each device.
2. bit1 = 0 to 7 and bit2 = 0 to 0FH.
3. Register-name-value1:
R0L=0 R0H=1 R1L=2 R1H=3 R2L=4 R2H=5 R3L=6 R3H=7
R4L=8 R4H=9 R5L=0AH R5H=0BH R6L=0CH R6H=0DH
R7L=0EH R7H=0FH
Register-name-vale2:
R0=0 R0=2 R2=4 R3=6 R4=8 R5=0AH R6=0CH R7=0EH
4. 0FFxxH denotes the address of an sfr or bsfr.
0xxxxH denotes the value of an expression.

Application Example

```
MOV1    CY, 0FE20H.3
AND1    CY, A.5
CLR1    P1.2
SET1    1+FE30H.3       ; Equals 0FE31H. 3.
SET1    0FE40H.4+2      ; Equals 0FE40H. 6.
```

2.5 Characteristics of Operands

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

For example, an instruction "MOV r1,#byte" functions to transfer the value indicated by "byte" to register "r1". In this case, because r1 is an 8-bit register, the size of the data "byte" to be transferred must be 8 bits or less.

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

So, when you describe an operand, attention must be paid to the following points:

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

2.5.1 Size and address range of operand value

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

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

These conditions are shown in Tables 2-15 and 2-16 below.

Table 2-15. Sizes and Address Ranges of Operand Values of Instructions

<table>
<tr><th rowspan="2">Operand representation format</th><th rowspan="2" colspan="2">Size &amp; address range of operand value</th><th rowspan="2">78K/0</th><th colspan="2">78K/I,II</th><th colspan="2">78K/III</th><th rowspan="2">78K/VI</th></tr>
<tr><th>112</th><th>Others</th><th>31X</th><th>Others</th></tr>
<tr><td rowspan="2">saddr</td><td colspan="2">0FE20H to 0FF1FH</td><td>o</td><td colspan="2"></td><td colspan="2">o</td><td></td></tr>
<tr><td colspan="2">0FE40H to 0FF1FH</td><td></td><td colspan="2"></td><td colspan="2">o</td><td></td></tr>
<tr><td>bsaddr</td><td colspan="2">0FC00H to 0FEFFH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td>saddrp</td><td colspan="2">Even value of 0FE20H to 0FF1FH</td><td>o</td><td colspan="2">o</td><td colspan="2">o</td><td></td></tr>
<tr><td>wsaddr</td><td colspan="2">Even value of 0FC00H to 0FF1EH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td>dsaddr</td><td colspan="2">Multiple of 4 of 0FC00H to 0FEFCH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td rowspan="5">addr16</td><td colspan="2">MOV 0H to
MOVW 0FFFFH</td><td>o</td><td colspan="2">o</td><td colspan="2">o</td><td></td></tr>
<tr><td rowspan="4">Other instruction</td><td>0H to 0FA7FH</td><td>o</td><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td>0H to 0FEFFH</td><td></td><td></td><td>o</td><td>o</td><td></td><td></td></tr>
<tr><td>0H to 0FDFFH</td><td></td><td colspan="2"></td><td></td><td>o</td><td></td></tr>
<tr><td>0H to 0FFFFH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td>addr13</td><td colspan="2">0H to 1FFFH</td><td></td><td>o</td><td></td><td colspan="2"></td><td></td></tr>
<tr><td>addr11</td><td colspan="2">800H to 0FFFH</td><td>o</td><td colspan="2">o</td><td colspan="2">o</td><td></td></tr>
<tr><td>addr8</td><td colspan="2">0H to 0FFH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td rowspan="2">addr5</td><td colspan="2">Even value of 40H to 7EH</td><td>o</td><td colspan="2">o</td><td colspan="2">o</td><td></td></tr>
<tr><td colspan="2">Even value of 8040H to 807EH</td><td></td><td colspan="2"></td><td colspan="2">o</td><td></td></tr>
<tr><td>byte</td><td colspan="2">8-bit value 0H to 0FFH</td><td>o</td><td colspan="2">o</td><td colspan="2">o</td><td>o</td></tr>
<tr><td>word</td><td colspan="2">16-bit value 0H to 0FFFFH</td><td>o</td><td colspan="2">o</td><td colspan="2">o</td><td>o</td></tr>
<tr><td>dword</td><td colspan="2">32-bit value 0H to 0FFFFFFFFH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td>bit</td><td colspan="2">3-bit value 0 to 7</td><td>o</td><td colspan="2">o</td><td colspan="2">o</td><td></td></tr>
<tr><td>bit3</td><td colspan="2">3-bit value 0 to 7</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td>bit4</td><td colspan="2">4-bit value 0H to 0FH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td rowspan="2">n</td><td colspan="2">2-bit value 0 to 3</td><td>o</td><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td colspan="2">3-bit value 0 to 7</td><td></td><td colspan="2">o</td><td colspan="2">o</td><td></td></tr>
<tr><td>n3</td><td colspan="2">3-bit value 0 to 7</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td>n4</td><td colspan="2">4-bit value 0H to 0FH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
<tr><td>n5</td><td colspan="2">5-bit value 0H to 1FH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>o</td></tr>
</table>

Table 2-16. Sizes and Address Ranges of Operand Values of Directives

<table>
<tr><th colspan="2" rowspan="2">Directive</th><th rowspan="2">Size &amp; address range of operand value</th><th rowspan="2">78K/0</th><th colspan="2">78K/I,II</th><th colspan="2">78K/III</th><th rowspan="2">78K/VI</th></tr>
<tr><th>112</th><th>Others</th><th>31X</th><th>Others</th></tr>
<tr><td>1</td><td>CSEG AT</td><td>0H to 0FEDDH</td><td>○</td><td colspan="2">○</td><td colspan="2">○</td><td>○</td></tr>
<tr><td></td><td>DESG AT</td><td>0H to 0FEFFH</td><td>○</td><td colspan="2">○</td><td colspan="2">○</td><td>○</td></tr>
<tr><td></td><td>BSEG AT</td><td>0H to 0FEFFH</td><td>○</td><td colspan="2"></td><td colspan="2"></td><td>○</td></tr>
<tr><td></td><td></td><td>0FE40H to 0FEFFH</td><td></td><td>○</td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td>0FE20H to 0FEFFH</td><td></td><td></td><td>○</td><td>○</td><td></td><td></td></tr>
<tr><td></td><td>ORG</td><td>0H to 0FFFfH</td><td>○</td><td colspan="2">○</td><td colspan="2">○</td><td>○</td></tr>
<tr><td>2</td><td>EQU</td><td>16-bit value
0H to 0FFFH</td><td>○</td><td colspan="2">○</td><td colspan="2">○</td><td>○</td></tr>
<tr><td></td><td>EQUD</td><td>32-bit value
0H to 0FFFFFFFFH</td><td></td><td colspan="2"></td><td colspan="2"></td><td>○</td></tr>
<tr><td></td><td>SET</td><td>16-bit value
0H to 0FFFFH</td><td>○</td><td colspan="2">○</td><td colspan="2">○</td><td>○</td></tr>
<tr><td>3</td><td>DB</td><td>8-bit value
0H to 0FFH</td><td>○</td><td colspan="2">○</td><td colspan="2">○</td><td>○</td></tr>
<tr><td></td><td>DW</td><td>16-bit value
0H to 0FFFFH</td><td>○</td><td colspan="2">○</td><td colspan="2">○</td><td>○</td></tr>
<tr><td></td><td>DS</td><td>16-bit value
0H to 0FFFFH</td><td>○</td><td colspan="2">○</td><td colspan="2">○</td><td>○</td></tr>
<tr><td>4</td><td>BR</td><td>0H to 0FEFFH</td><td>○</td><td colspan="2">○</td><td colspan="2">○</td><td>○</td></tr>
<tr><td>5</td><td>RSS</td><td>1-bit value
0 or 1</td><td></td><td colspan="2"></td><td colspan="2">○</td><td></td></tr>
</table>

Types of directives:
1 Segment Definition Directives
2 Symbol Defintion Directives
3 Memory Initialization and Area Reservation Directives
4 Automaic Brach Instruction Selection Directive
5 General-purpose Register Selection Directive (applicable to the 78K/III only)

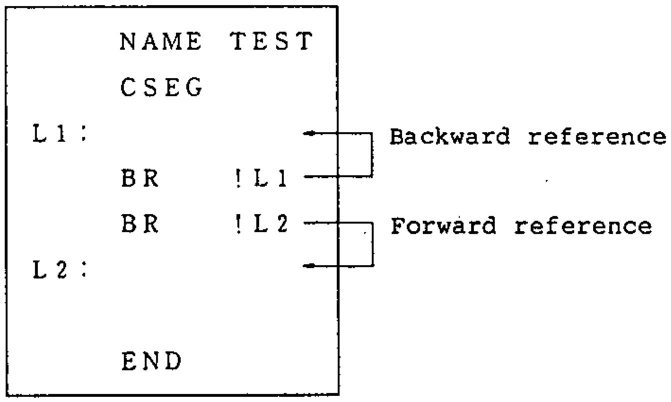
2.5.2 Symbol attributes and relocation attributes of operands

When describing a name, label, or $ (location counter) as the operand of an instruction, the name or label may or may not be described as an operand depending on the symbol attribute and relocation attribute of the operand as the term in an expression (see 2.3.2, "Restrictions on operations") or the reference direction of the name or label.

Names and labels are referenced in two directions: Backward and Forward.

- o Backward reference .... Name or label to be referenced as an operand has been defined in a previous line.
- o Forward reference ..... Name or label to be referenced as an operand has been defined in a subsequent line.

Example:



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

        END
```

These attributes of the operands of instructions and directives are shown in Tables 2-17 and 2-18 below.

Table 2-17. Attributes of Instruction Operands

Symbol attribute	NUMBER		ADDRESS				NUMBER or ADDRESS		Special function register (sfr name)	DNUMBER					
Relocation attribute	Absolute term		Absolute term		Relocatable term		External ref. term			Absolute term					
Reference direction / Operand format	BW	FW	BW	FW	BW	FW	BW	FW		BW	FW	78K/0	78K/I	78K/II,III	78K/VI
sfr	o Cf.1	-	-	-	-	-	-	-	o Cf.2,3	-	-	o		o	
sfrp	-	-	-	-	-	-	-	-	o Cf.2,4	-	-	o	o	o	
bsfr	-	-	-	-	-	-	-	-	o Cf.3	-	-				o
wsfr	-	-	-	-	-	-	-	-	o Cf.4	-	-				o
saddr	o	o	o	o	o	o	o	o	o Cf.2,5	-	-	o	o	o	
saddrp	o	o	o	o	o	o	o	o	o Cf.2,6			o	o	o	
bsaddr	o	o	o	o	o	o	o	o	-	-	-				o
wsaddr	o	o	o	o	o	o	o	o	-	-	-				o
dsaddr	o	o	o	o	o	o	o	o	-	-	-				o
addr16 Cf.1	o	o	o	o	o	o	o	o	-	-	-	o	o	o	o
addr11	o	o	o	o	o	o	o	o	-	-	-	o	o	o	
addr8	o	o	o	o	o	o	o	o	-	-	-				o
addr5	o	o	o	o	o	o	o	o	-	-	-	o	o	o	
dword	-	-	-	-	-	-	-	-	-	o	-				o
word	o	o	o	o	o	o	o	o	-	-	-	o	o	o	o
byte	o	o	o	o	o	o	o	o	-	-	-	o	o	o	o
bit	o	o	-	-	-	-	-	-	-	-	-	o	o	o	
bit3/bit4	o	o	-	-	-	-	-	-	-	-	-				o
bit3/bit4	o	o	-	-	-	-	-	-	-	-	-	o	o	o	
n3/n4/n5	o	o	-	-	-	-	-	-	-	-	-				o

BW ... Backward reference   FW ... Forward reference
o ... Can be described.   - ... Cannot be described.

Notes: 1. The address of an external area to be accessed must be described with an absolute expression.
2. If an sfr reserved word in the saddr area has been described for an instruction in which a combination of sfr/sfrp changed from saddr/saddrp exists as operands, codes will be output as saddr/saddrp.
3. sfr reserved word that allows 8-bit accessing
4. sfrp reserved word that aloows 16-bit accessing
5. sfr reserved word for asddr area
6. sfrp reserved word for saddr area

Table 2-18. Attributes of Directive Operands

Symbol attribute		NUMBER		ADDRESS						BIT					
Relocation attribute		Abs. term		Abs. term		Rel. term		Ext. term		Abs. ref.		Rel. term		Ext. term	
Reference direction Directive		BW	FW	BW	FW	BW	FW	BW	FW	BW	FW	BW	FW	BW	FW
ORG		o Cf1	-	-	-	-	-	-	-	-	-	-	-	-	-
EQU		o	-	o	-	o Cf2	-	-	-	o	-	o Cf2	-	-	-
SET		o Cf1	-	-	-	-	-	-	-	-	-	-	-	-	-
DB	Size	o Cf1	-	-	-	-	-	-	-	-	-	-	-	-	-
	Initial value	o	o	o	o	o	o	o	o	-	-	-	-	-	-
DW	Size	o Cf1	-	-	-	-	-	-	-	-	-	-	-	-	-
	Initial value	o	o	o	o	o	o	o	o	-	-	-	-	-	-
DS		o	-	-	-	-	-	-	-	-	-	-	-	-	-
BR		o	o	o	o	o	o	o	o	-	-	-	-	-	-
RSS		o Cf1	o	-	-	-	-	-	-	-	-	-	-	-	-

Abs. term ... Absolute term   Rel. term ... Relocatable term
Ext. term ... External reference term
BW ... Backward reference   FW ... Forward reference
o ... Can be described.   - ... Cannot be described.

Notes: 1. Only an absolute expression can be described.
2. A term created by the HIGH or LOW operator which has a relocatable term as its operand is not allowed.
3. An error will result if an expression includes one of the following four attribute combinations and the result of the operation is likely to be affected by the optimization:
   - o ADDRESS attribute - ADDRESS attribute
   - o ADDRESS attribute relational operator ADDRESS attribute
   - o HIGH operator absolute ADDRESS attribute
   - o LOW operator absolute ADDRESS attribute

CHAPTER 3. DIRECTIVES

3.1 Overview of Directives

Directives are described in a source program just the same as ordinary instructions. Directives are pseudoinstructions in a program which give the assembler processor various instructions necessary for this package to perform a series of processes. Instructions will be translated into object codes (i.e., machine language), but directives will not, as a rule, be converted into object codes.

Directives are available in nine types as listed in Table 3-1 and have the following major functions:

- o Facilitate description of source programs.
- o Initialize memory and reserve memory areas.
- o Provide the information required for the assembler and linker to perform their intended processing.

Table 3-1. List of Directives

No.	Type of directive	Directives
1	Segment definition directives	CSEG, DSEG, BSEG, ORG, ENDS
2	Symbol definition directives	EQU, EQUD (applicable to 78K/VI only), SET
3	Memory initialization/ area reservation directives	DB, DW, DS, DBIT
4	Linkage directives	PUBLIC, EXTRN, EXTBIT
5	Object module declaration directive	NAME
6	Automatic branch instruction selection directive	BR
7	General register selection directive	RSS (applicable to 78K/III only)
8	Macro directives	MACRO, LOCAL, REPT, IRP, EXITM, ENDM
9	Assembly termination directive	END

A detailed description of each of these directives will be provided in the following sections.

In the description format of each directive, "[   ]" indicates that the parameter in braces may be omitted from specification and "..." indicates the repetition of description in the same format.

For example, if the description format reads:

```
[(A)][B[, ...]]
```

you may describe parameter(s) in any of the following three formats:

- o A
- o A B, B, B
- o B, B

3.2 Segment Definition Directives

A source module is described in units of segments.
Segment definition directives are used to define these segments.
Segments are divided into the following four types:

- o Code segment
- o Data segment
- o Bit segment
- o Absolute segment

To which address range in memory each segment will be located is determined by the type of segment.
Table 3-2 shows the method of defining each segment and the memory address area to be allocated to each segment.

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

Type of segment	Method of definition	Memory address area to be allocated to each segment
Code segment	CSEG directive	Within the internal or external ROM area
Data segment	DSEG directive	Within the internal or external RAM area
Bit segment	BSEG directive	Within the saddr area in the internal RAM
Absolute segment	Specifies location address (AT location address) to relocation attribute with CSEG, DSEG, or BSEG directive	Specified address

If the user wishes to determine the memory allocation to a segment, describe (define) the segment as an absolute segment.
An example of memory allocation to segments is shown in Fig. 3-1.

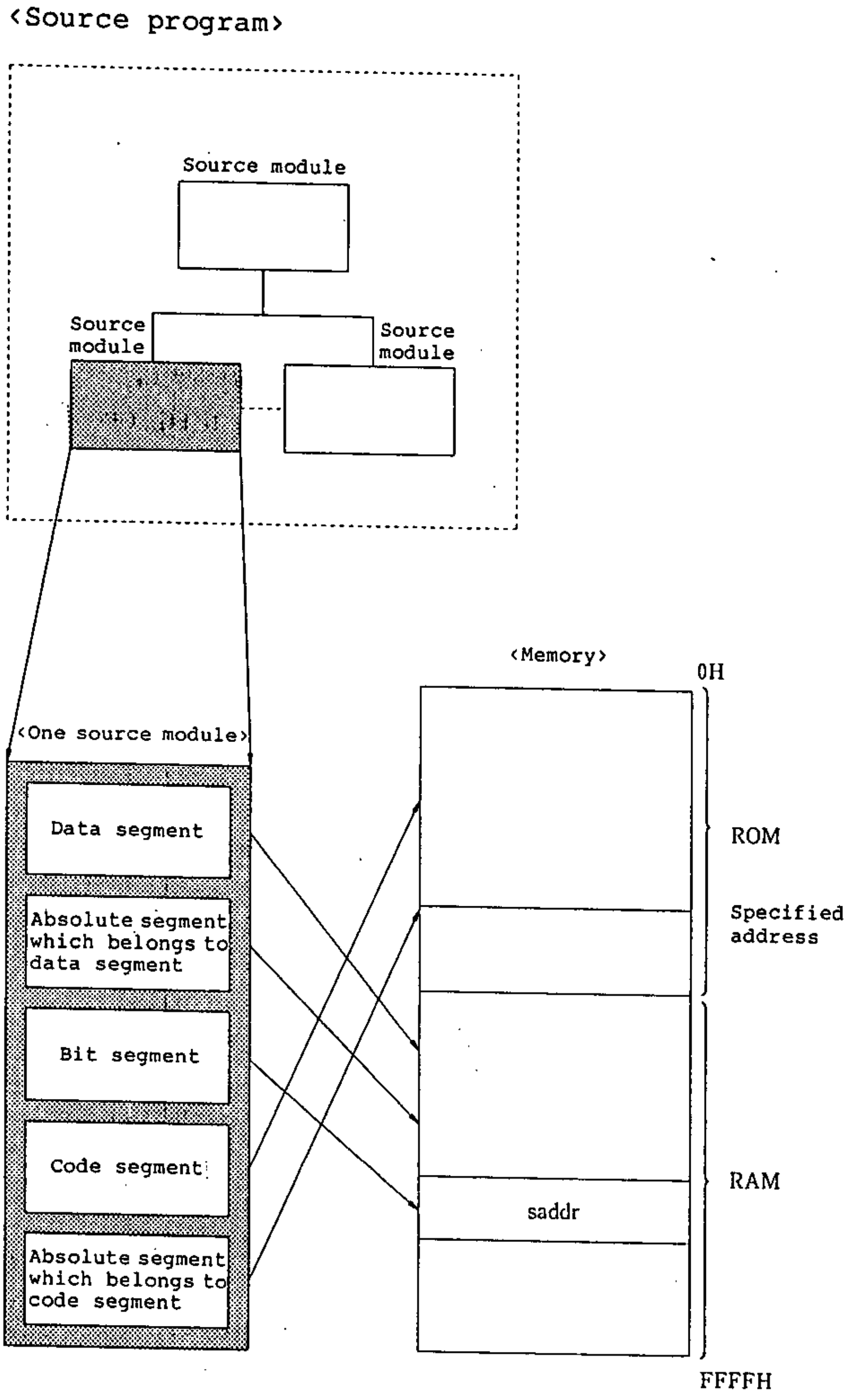


Fig. 3-1. Memory Allocation to Segments

(1) CSEG (code segment)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[segment name]	CSEG	[relocation attribute]	[;comment]

Function

- o The CSEG directive indicates to the assembler the start of a code segment.
- o All instructions described after this directive until the re-appearance of any segment definition directive (CSEG, DSEG, BSEG, ORG, or ENDS) in the source module will belong to the code segment and will be located within the ROM address area upon conversion into machine language.

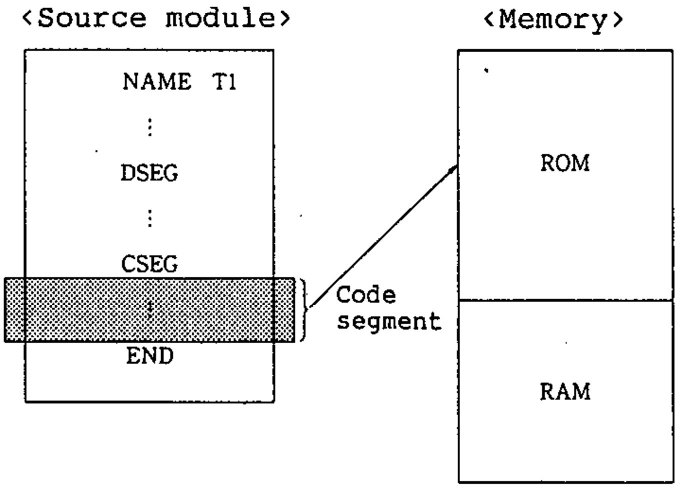


Fig. 3-2. Relocation of Code Segment

Use

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

Explanation

- The start address of a code segment can be specified with the ORG directive. It can also be specified by describing the relocation attribute "AT" followed by an absolute expression in the Operand field of the CSEG directive.
- A relocation attribute defines a range of location addresses for a code segment. There are seven types of relocation attributes available for code segments as shown in Table 3-3.

Table 3-3. Relocation Attributes of CSEG

Relocation attribute	Description format	Explanation	78K/0	78K/I	78K/II	78K/III	78K/VI
CALLT0	CALLT0	Instructs the assembler to locate the specified segment so that its first address becomes a multiple of 2 within addresses 0040H to 007FH. Specify this relocation attribute for a code segment which defines the entry address of a subroutine to be called with the one-byte "CALLT" instruction (i.e., if CCW TPF=0)	o	o	o	o	
CALLT1	CALLT1	Instructs the assembler to locate the specified segment so that its first address becomes a multiple of 2 within addresses 2040H to 807FH. Specify this relocation attribute for a code segment which defines the entry address of a subroutine to be called with the one-byte "CALLT" instruction (i.e., if CCW TPF=1)				o	
TABLE	TABLE	Instructs the assembler to locate the specified segment so that its first address becomes a multiple of 2 within addresses 0050H to 00FFH.					o
FIXED	FIXED	Instructs the assembler to locate the specified segment within addresses 0800H to 0FFFH. Specify this relocation attribute for a code segment which defines a subroutine to be called with the 2-byte "CALLF" instruction.	o	o	o	o	
AT	AT absolute expression	Instructs the assembler to locate the specified segment to an absolute address (within 0000H to FEFFH).	o	o	o	o	o

Table 3-3. Relocation Attributes of CSEG (contd)

Relocation attribute	Description format	Explanation	78K/0	78K/I	78K/II	78K/III	78K/VI
UNIT	UNIT (default value)	Instructs the assembler to locate the specified segment to any address within the following range: With 78K/0: 0080H to FA7FH With 78K/I,III: 0000H to FEFFH With 78K/II: 0080H to FEFFH With 78K/VI: 0100H to the last address of ROM area	o	o	o	o	o
PAGE	PAGE	Instructs the assembler so that the high-order 8 bits are located within the same address range.			o		

o: Applicable Blank: Not applicable

NOTE: "CCW TPF" refers to the table position flag (TPF) of the CPU control word (CCW) which specifies the location of a vector table to be referenced by the CALLT instruction or or an interrupt request. According to the value of this flag, the vector table location is switched and selects the address indicated by the vector table. CCW is mapped to the address 0FE4EH of the SFR (special function register) area.

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

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

Table 3-4. Default Segment Names of CSEG

Relocation attribute	Default segment name
CALLT0 (applicable to all devices except 78K/VI)	?CSEGT0
CALLT1 (applicable to 78K/III only)	?CSEGT1
FIXED	?CSEGFX
UNIT	?CSEG (default value)
AT	Segment name cannot be omitted. ?CSEG is assumed from default relocation attribute "UNIT".

- If two or more code segments with the same relocation attribute (except AT) exist in a source module, these code segments may have the same segment name. These same named code segments will be processed as a single code segment within the assembler.
- An error will result if the same named segments differ in their relocation attributes. Therefore, the number of the same named segments for each relocation attribute is one.
- The same named code segments in two or more different modules will be combined into a single code segment at linkage time.
- No segment name can be referenced as a symbol.
- The total number of segments that can be output by the assembler is as follows:

With 78K/0, II, and VI:
100 segments including code, data, and bit segments. The same named segments are counted as one.
With 78K/I and III:
80 segments under different segment names with the exception of segments defined with the ORG directive. The same named segments are counted as one.

Application Examples

Example 1

```
         NAME     SAMP1
         CSEG     CALLT0      ;(1)
TLAB1:   DW       LAB1

         CSEG                 ;(2)
           :
         CLR1     CCW.1       ;(3)
         CALLT    [TLAB1]     ;(4)

         CSEG                 ;(5)
LAB1:      :

         END
```

(1) Within this code segment, the entry address of a subroutine to be called by the CALLT instruction if TPF = 0 is defined. Therefore, relocation attribute "CALLT0" must be specified for this code segment.
(2) Within this code segment, instructions which may be located to any locations in the ROM area are described. Therefore, no relocation attribute will be specified for this code segment.
(3) Clears TPF (Bit 1 of CCW). (TPF = 0)

(4) Label "TLAB1" is described to indicate the address in which the entry address of the subroutine is stored.

(5) In this code segment, the subroutine to be called by the CALLT instruction in (4) is defined.

Example 2

```
        NAME   SAMP2
        CSEG   CALLT1     ;(1)
TLAB2:  DW     LAB2

        CSEG              ;(2)
          ⋮
        SET1   CCW.1      ;(3)
        CALLT  [TLAB2]    ;(4)

        CSEG              ;(5)
LAB2:     ⋮

        END
```

(1) Within this code segment, the entry address of a subroutine to be called by the CALLT instruction if TPF = 1 is defined. Therefore, relocation attribute "CALLT1" must be specified for this code segment.

(2) Within this code segment, instructions which may be located to any locations in the ROM area are described. Therefore, no relocation attribute will be specified for this code segment.

(3) Sets TPF (Bit 1 of CCW). (TPF = 1)

(4) Label "TLAB2" is described to indicate the address in which the entry address of the subroutine is stored.

(5) In this code segment, the subroutine to be called by the CALLT instruction in (4) is defined.

Example 3

```
        NAME    SAMP3
        CSEG    FIXED    ;(1)
SUB1:     :

        CSEG
          :
        CALLF   !SUB1    ;(2)
          :
        END
```

(1) Within this code segment, the entry address of a subroutine to be called by the CALLF instruction is defined. Therefore, relocation attribute "FIXED" must be specified for this code segment.

(2) Label "SUB1" is described as the operand of the CALLF instruction to indicate the address in which the entry address of the subroutine is stored.

Example 4

```
        NAME    SAMP4
SN41    CSEG             ;(1)
          :
SN42:   CSEG             ;(2)
          :
        END
```

This description is erroneous.

(1) A code segment is defined without relocation attribute specification (segment type "CSEG"). Segment name is "SN41".

(2) A code segment without specified relocation attribute (segment type "CSEG") has already been defined in (1) above. Therefore, this description results in an error. Change the segment name to "SN41".

Example 5

```
        NAME    SAMP5
SN5     CSEG    AT 1000H;(1)
          :
SN5     CSEG            ;(2)
          :
        END
```

This description is erroneous.

(1) The location of a code segment is specified as an absolute segment with start address "1000H".

(2) The same named segment is not allowed with the AT relocation attribute.

Example 6

<Module 1>

```
        NAME    SAMP61
SN6     CSEG            ;(1)
          :
        END
```

<Module 2>

```
        NAME    SAMP62
SN6     CSEG            ;(2)
          :
        END
```

(1) and (2)

Code segment "SN6" defined in (1) in module 1 becomes the same in segment name and segment type as the code segment defined in (2) in module 2. Therefore, these two code segments will be processed as a single code segment at linkage time.

(2) DSEG (data segment)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[segment name]	DSEG	[relocation attribute]	[;comment]

Function

- o The DSEG directive indicates to the assembler the start of a data segment.
- o Memory areas defined by the DS directive after this directive until the re-appearance of any segment definition directive (CSEG, DSEG, BSEG, ORG, or ENDS) in the source module will belong to the data segment and will be finally allocated within the RAM address area.

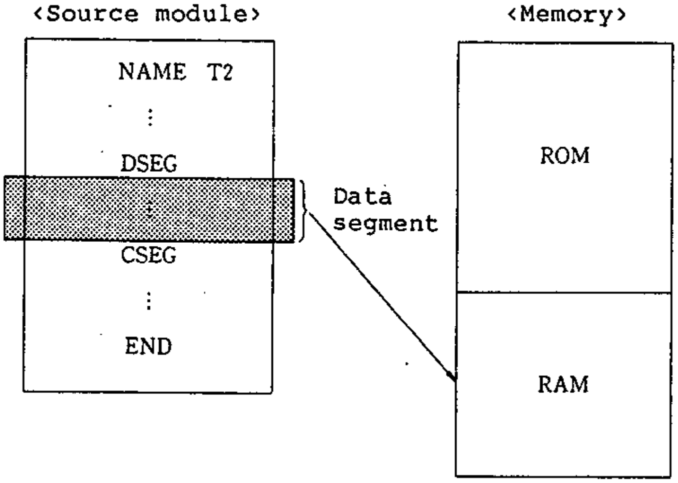


Fig. 3-3. Relocation of Data Segment

Use

- The DS directive is mainly described in the data segment defined by the DSEG directive. Data segments will be located within the RAM area. Therefore, no instructions can be described in any data segment.
- In a data segment, the DS directive must be described to reserve a RAM work area to be used by the program and a label must be given to the address of each work area. When describing a source program, this label is used. Each area reserved as a data segment will be allocated by the linker so that it does not overlap with any other work areas on the RAM (stack area, general-purpose register area, and work areas defined by other modules).

Explanation

- The start address of a data segment can be specified with the ORG directive. It can also be specified by describing the relocation attribute "AT" followed by an absolute expression in the Operand field of the DSEG directive.
- A relocation attribute defines a range of location addresses for a data segment. There are four types of relocation attributes available for data segments as shown in Table 3-5.

Table 3-5. Relocation Attributes of DSEG

Relocation attribute	Description format	Explanation	78K/0	78K/I	78K/II	78K/III	78K/VI
SADDR	SADDR	Instructs the assembler to locate the specified segment to the following addresses within the saddr area. With 78K/0, I, and III: FE20H to FEFFH With uPD78112: FE40H to FEFFH With 78K/VI: FCFFH to FEFFH	o	o	o	o	o
SADDRP	SADDRP	Instructs the assembler to locate the specified segment to addresses FE20H to FEFFH within the saddr area so that its first address becomes a multiple of 2.			o		
WSADDRP	WSADDRP	Instructs the assembler to locate the specified segment to addresses FC00H to FEFFH within the saddr area so that its first address becomes a multiple of 2.					o
IHRAM	IHRAM	Instructs the assembler to locate the specified segment within the high-speed RAM area.	o				
AT	AT absolute expression	Instructs the assembler to locate the specified segment to an absolute address (within 0000H to FEFFH).	o	o	o	o	o
UNIT	UNIT (default value)	Instructs the assembler to locate the specified segment to any address within the memory area name RAM.	o	o	o	o	o
PAGE	PAGE	Instructs the assembler so that the the high-order 8 bits are located within the same address range.			o		

o: Applicable Blank: Not applicable

o If no relocation attribute is specified for the data segment, "UNIT" is assumed to have been specified.

o If a relocation attribute other than those listed in Table 3-5 is specified, the assembler will output an error message and assume that "UNIT" has been specified. An error will result if the size of each data segment exceeds the size of the area specified by its relocation attribute.

- o If the absolute expression specified with the AT relocation attribute is illegal, the assembler will output an error message and continue processing by assuming the value of the expression to be "0".
- o By describing a segment name in the Symbol field of the DSEG directive, the data segment can be named.

  If no segment name is specified for a data segment, the assembler will automatically give a segment name to the data segment. The default segment names of data segments are shown in Table 3-6.

Table 3-6. Default Segment Names of DSEG

Relocation attribute	Default segment name
SADDR	?DSEGS
SADDRP (applicable to 78K/III only)	?DSEGSP
WSADDR (applicable to 78K/VI only	?DSEGWS
DSADDR (applicable to 78K/VI only	?DSEGDS
IHRAM (applicable to 78K/0 only)	?DSEG
UNIT	?DSEG (default value)
AT	Segment name cannot be omitted. ?DSEG is assumed from default relocation attribute "UNIT".

- o If two or more data segments with the same relocation attribute (except AT) exist in a source module, these data segments may have the same segment name. These same named data segments will be processed as a single data segment within the assembler.
- o If the relocation attribute is SADDRP or WSADDR, the specified segment will be located so that its first address (the address immediately after the described DSEG directive) becomes a multiple of 2. With the DSADDR relocation attribute, the specified segment will be located so that its first address becomes a multiple of 4. The assembler will provide a gap of 1 to 3 bytes if necessary.

- o An error will result if the same named segments differ in their relocation attributes. Therefore, the number of the same named segments for each relocation attribute is one.
- o The same named data segments in two or more different modules will be combined into a single data segment at linkage time.
- o No segment name can be referenced as a symbol.
- o The total number of segments that can be output by the assembler is as follows:
  With 78K/0, II, and VI:
  100 segments including data, code, and bit segments. The same named segments are counted as one.
  With 78K/I and III:
  80 segments under different segment names with the exception of segments defined with the ORG directive. The same named segments are counted as one.

Application Examples

Example 1

```
        NAME  SAMP1
        DSEG               ;(1)
WORK1:  DS    1
WORK2:  DS    2
        CSEG
          :
        MOV   A,!WORK1     ;(2)
        MOV   A,WORK1      ;(3)  <--- This description is
          :                           erroneous.
        MOVW  DE,#WORK2    ;(4)
        MOVW  AX,[DE]
          :
        MOVW  AX,WORK2     ;(5)  <--- This description is
          :                           erroneous.
        END
```

(1) The start of a data segment is defined with the DSEG directive. Because its relocation attribute is omitted, "UNIT" is assumed.

(2) This description corresponds to "MOV A,!addr16".

(3) This description corresponds to "MOV A,saddr". Relocatable label "WORK1" cannot be described as "saddr". Therefore, an error will occur as a result of this description.

(4) This description corresponds to "MOVW rp1,#word".

(5) This description corresponds to "MOVW AX,saddrp". Relocatable label "WORK2" cannot be described as "saddrp". Therefore, an error will result as a result of this description.

Example 2

```
        NAME  SAMP2
DATA1   DSEG                 ;(1)
          :
        CSEG
          :
DATA1   DSEG                 ;(2)
          :
        END
```

(1) A data segment with segment name "DATA1" is defined with the DSEG directive.

(2) This segment will be processed as a continuous segment which follows the data segment defined in (1) above. The first address of the segment defined in (2) will be the address next to the last address of the segment defined in (1) above.

(3) BSEG (bit segment)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[segment name]	BSEG	[relocation attribute]	[;comment]

Function

- o The BSEG directive indicates to the assembler the start of a bit segment.
- o A bit segment is a segment which defines the RAM addresses to be used in the source module.
- o Memory areas defined by the DBIT directive after this directive until the re-appearance of any segment definition directive (CSEG, DSEG, BSEG, ORG, or ENDS) in the source module will belong to the bit segment and will be finally allocated to addresses within the saddr area (the RAM area with the 78K/VI).

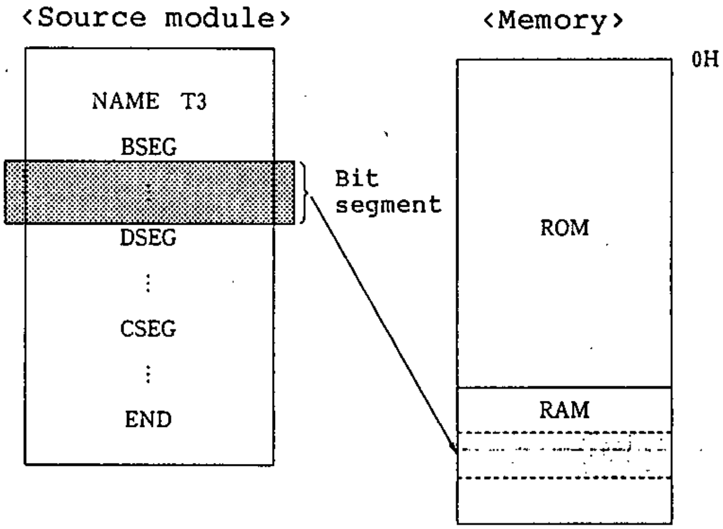


Fig. 3-4. Relocation of Bit Segment

Use

- o Describe the DBIT directive in the bit segment defined by the BSEG directive. (See Application Examples 1 and 2.)
- o No instructions can be described in any bit segment.

Explanation

- o The start address of a bit segment can be specified by describing the relocation attribute "AT" followed by an absolute expression in the Operand field of the BSEG directive.
- o A relocation attribute defines a range of location addresses for a bit segment. There are three types of relocation attributes available for bit segments as shown in Table 3-7.

Table 3-7. Relocation Attributes of BSEG

Relocation attribute	Description format	Explanation	78K/0	78K/I	78K/II	78K/III	78K/VI
SADDR	SADDR	Instructs the assembler to locate the specified segment to addresses FC00H to FEFFH within the saddr area. The beginning of the segment is located at the 0th bit.					o
AT	AT absolute expression	Instructs the assembler to locate the beginning of the specified segment to the 0th bit of an absolute address within the following address range: With 78K/0 and VI: 0000H to FEFFH With 78K/I, II and III: FE20H to FEFFH With uPD78112: FE40H to FEFFH	o	o	o	o	o
UNIT	UNIT (default value)	Instructs the assembler to locate the specified segment to any address within the following address range: With 78K/0, I, II, and III: FE20H to FEFFH With uPD78112: FE40H to FEFFH With 78K/VI: Default RAM area	o	o	o	o	o

o: Applicable Blank: Not applicable

- o If no relocation attribute is specified for the bit segment, "UNIT" is assumed to have been specified.
- o If a relocation attribute other than those listed in Table 3-7 is specified, the assembler will output an error message and assume that "UNIT" has been specified. An error will result if the size of each bit segment exceeds the size of the area specified by its relocation attribute.
- o In both the assembler and linker, the location counter in a bit segment is displayed in the form "xxxx. b". (The byte address is hexadecimal 4 digits and the bit position is hexadecimal 1 digit (0 to 7).)

With absolute bit segment

Byte address	0	1	2	3	4	5	6	7 ←Bit position
0FE20H	①	②	③	④	⑤	⑥	⑦	⑧
0FE21H	⑨	⑩	⑪	⑫	⑬	⑭	⑮	⑯
:								

Location counter

①	0FE20H.0	⑨	0FE21H.0
②	0FE20H.1	⑩	0FE21H.1
③	0FE20H.2	⑪	0FE21H.2
④	0FE20H.3	⑫	0FE21H.3
⑤	0FE20H.4	⑬	0FE21H.4
⑥	0FE20H.5	⑭	0FE21H.5
⑦	0FE20H.6	⑮	0FE21H.6
⑧	0FE20H.7	⑯	0FE21H.7

With relocatable bit segment

Byte address	0	1	2	3	4	5	6	7 ←Bit position
0H	①	②	③	④	⑤	⑥	⑦	⑧
1H	⑨	⑩	⑪	⑫	⑬	⑭	⑮	⑯
:								

Location counter

①	0H.0	⑨	1H.0
②	0H.1	⑩	1H.1
③	0H.2	⑪	1H.2
④	0H.3	⑫	1H.3
⑤	0H.4	⑬	1H.4
⑥	0H.5	⑭	1H.5
⑦	0H.6	⑮	1H.6
⑧	0H.7	⑯	1H.7

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

In a symbol table to be output by the object converter, a bit offset from the beginning of an area in which a bit has been defined is displayed.

Symbol value	Bit offset
FE20H.0	0000
FE20H.1	0001
FE20H.2	0002
⋮	⋮
FE20H.7	0007
FE21H.0	0008
FE21H.1	0009
⋮	⋮
FE80H.0	0300
⋮	⋮

- o If the absolute expression specified with the AT relocation attribute is illegal, the assembler will output an error message and continue processing by assuming the value of the expression to be "0".
- o By describing a segment name in the Symbol field of the BSEG directive, the bit segment can be named.
  If no segment name is specified for a bit segment, the assembler will automatically give a segment name to the bit segment. The default segment names of bit segments are shown in Table 3-8.

Table 3-8. Default Segment Names of BSEG

Relocation attribute	Default segment name
SADDR (applicable to 78K/VI only)	?BSEGS
UNIT	?BSEG (default value)
AT	Segment name cannot be omitted. ?BSEG is assumed from default relocation attribute "UNIT".

- o If two or more bit segments with the same relocation attribute (except AT) exist in a source module, these bit segments may have the same segment name. These same named bit segments will be processed as a single bit segment within the assembler.
- o An error will result if the same named segments differ in their relocation attributes. Therefore, the number of the same named segments for each relocation attribute is one.
- o The same named data segments in two or more different modules will be combined into a single data segment at linkage time. This linking is performed in units of bits.
- o No segment name can be referenced as a symbol.
- o Only the DBIT, EQU, SET, PUBLIC, EXTBIT, EXTRN, MACRO, REPT, IRP, and ENDM directives, macrodefinitions and macro references can be described in a bit segment. An error will result if any instruction other than the above is described.
- o The total number of segments that can be output by the assembler is as follows:

  With 78K/0, II, and VI:

  100 segments including data, code, and bit segments. The same named segments are counted as one.

  With 78K/I and III:

  80 segments under different segment names with the exception of segments defined with the ORG directive. The same named segments are counted as one.

Application Examples

Example 1

```
        NAME    SAMP1
        BSEG                ;(1)
B1      DBIT
B2      DBIT
B3      DBIT

        CSEG
        MOV1    CY, B1      ;(2)
          :
        AND1    CY, B2      ;(3)
          :
        END
```

(1) A bit segment is defined with the BSEG directive. Because its relocation attribute is omitted, the relocation attribute "UNIT and the segment name "?BSEG" are assumed. In each bit segment, a bit work area is defined for each bit with the DBIT directive. A bit segment should be described at the early part of the module body.

(2) This description corresponds to "MOV1 CY,saddr.bit".

(3) This description corresponds to "AND1 CY,saddr.bit".

Example 2

```
        NAME    SAMP2

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

        BSEG
FLAG2   DBIT                    ;(2)

        CSEG
        MOV1    CY, FLAG0       ;(3)
          :
        MOV1    CY, FLAG2       ;(4)
          :
        END
```

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

(2) Bit address FLAG2 defined in the bit segment is located without consideration to any byte address boundary.

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

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

Example 3

```
        NAME    SAMP3
B1      BSEG    AT 0FE20H     ;(1)
          :
        END
```

(1) A bit segment is located as an absolute segment starting from address 0FE20H.

(4) ORG (origin)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[segment name]	ORG	[absolute expression]	[;comment]

Function

- o The ORG directive sets the value of the expression specified by its operand in the location counter.
- o Instructions described or memory areas reserved after this directive until the re-appearance of any segment definition directive (CSEG, DSEG, BSEG, ORG, or ENDS) in the source module will belong to the absolute segment and will be allocated beginning with the address specified in the operand of this directive.

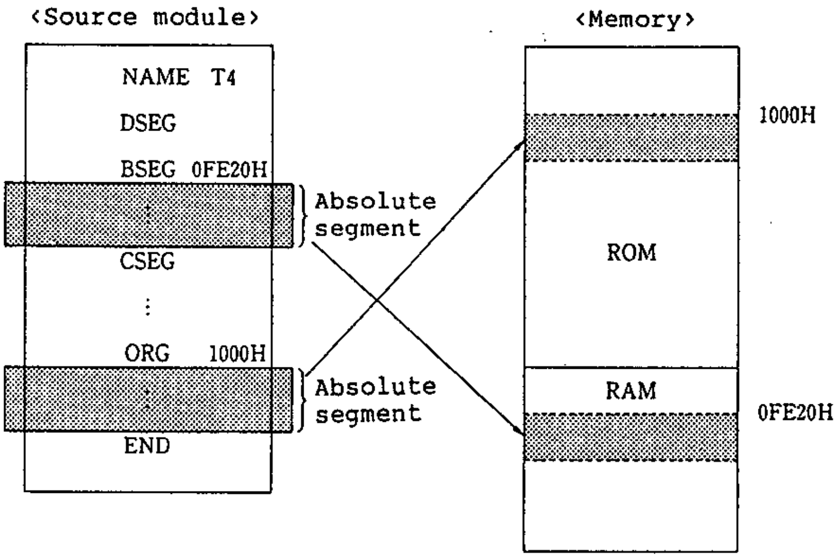


Fig. 3-5. Relocation of Absolute Segment

Use

Specify the ORG directive to start memory allocation to a code segment or data segment from a specific address.

Explanation

- The absolute segment defined with the ORG directive belongs to the code segment or data segment defined with the CSEG or DSEG immediately before the ORG directive.

  Within an absolute segment which belongs to a data segment, no instructions can be described.

  An absolute segment which belongs to a bit segment cannot be described with the ORG directive.
- The number of times that the ORG directives can be described per source module is as follows:

  With 78K/0, II, and VI:

  No specific restriction (provided the total number of segments must not exceed 100)

  With 78K/I and III: Up to 20 times
- The code segment or data segment defined with the ORG directive is interpreted as a code segment or data segment which has the relocation attribute AT.
- By describing a segment name in the Symbol field of the ORG directive, the absolute segment can be named. The maximum number of characters that can be recognized as a segment name is 8.
- If no segment name is specified for an absolute segment, the assembler will automatically give a default segment name as follows:

  With 78K/0, II and VI:

  "?A00xxxx" will be given to the absolute segment (where "xxxx" indicates the start address of the segment specified by the operand).

  With 78K/I and III:

  "?ASEGn" will be given to the absolute segment (where n is a value in the range of 1 to 20 and is given in the order of absolute segment description within the source module).
- If neither CSEG nor DSEG directive has been described before the ORG directive, the absolute segment defined by the ORG directive is interpreted as an absolute segment in a code segment.

- o If a name or label is to be described as the operand of the ORG directive, the name or label must be an absolute term which has already been defined in the source module.
- o No segment name can be referenced as a symbol.

Application Examples

Example 1

```
          NAME   SAMP1
          DSEG
SADR      ORG    0FE20H        ;(1)
SADR1:    DS     1
SADR2:    DS     1
SADR3:    DS     2
           :
MAIN0     ORG    100H
          MOV    A, SADR1      ;(2)
           :
          CSEG                 ;(3)
MAIN1     ORG    1000H         ;(4)
          MOV    A, SADR2
          MOVW   AX, SADR3
           :
          END
```

This description is erroneous.

(1) An absolute segment which belongs to a data segment is defined. This absolute segment will be located to the short direct addressing area which starts from address "FE20H".

(2) Within an absolute segment which belongs to a data segment, no instruction can be described.

(3) This directive declares the start of a code segment.

(4) This absolute segment will be located to an area which starts from address "1000H".

Example 2

```
NAME  SAMP2
ORG   0H          ;(1)
:
CSEG              ;(2)
ORG   800H        ;(3)
:
ORG   1000H       ;(4)
:
END
```

(1) This absolute segment belongs to a code segment and will be located to an area which starts from address "0H".

(2) This directive declares the start of a code segment.

(3) This absolute segment will be located to an area which starts from address "800H".

(4) This absolute segment will be located to an area which starts from address "1000H".

Three absolute segments have been defined without segment name in (1), (3), and (4) above. Therefore, the assembler will automatically give absolute segment names ?ASEG1, ?ASEG2, and ?ASEG3, respectively, to these segments in the order of their definition.

(5) ENDS (end of segment)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
None	ENDS	None	[;comment]

Function

The ENDS directive indicates the end of the relocatable segment defined by the CSEG, DSEG, or BSEG directive.

Use

The ENDS directive is used in pairs with the CSEG, DSEG, or BSEG directive in a source module.

Explanation

- o The ENDS directive indicates the end of each segment in source module description.
- o After the ENDS directive has been described, only a comment statement can be described before the next segment definition directive (CSEG, DSEG, BSEG, or ORG) is described.
- o Description of the ENDS directive may be omitted.

Application Example

```
            NAME      SAMP1
            BSEG
              :
            ENDS                ;(1)
;data       segment
            DSEG                ;(2)
              :
            ENDS                ;(3)
WORK:       DS        1         ;(4)  ←—This description is
            CSEG                ;(5)     erroneous.
              :
            ENDS                ;(6)
            END
```

(1) This directive indicates the end of a bit segment.

(2) This directive indicates the start of a data segment. Only comments can be described between (1) and (2).

(3) This directive indicates the end of the data segment.

(4) Only a comment statement can be described between the ENDS directive and the CSEG directive in (5) below.

(5) This directive indicates the start of a code segment.

(6) This directive indicates the end of the code segment.

3.3 Symbol Definition Directives

Symbol definition directives assign names to numerical data to be used for describing a source module. By these names, the meaning of each data value becomes clear and you may easily understand the contents of the source module.

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

Three directives are available for symbol definition: EQU, EQUD, and SET.

(1) EQU (equate)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
name	EQU	expression	[;comment]

Function

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

Use

Define a numerical data to be used in the source module as a name with the EQU directive and describe it in the operand of an instruction in place of the numerical data.
Numerical data to be frequently used in the source module should be defined as names. By so doing, if you must change a data value in the source module, all you need to do is change the operand value of the name. (See Application Example 1.)

Explanation

o When a name or label is to be described in the operand of the EQU directive, use the name or label which has already been defined in the source module.
No external reference term can be described as the operand of this directive.

o If an expression includes a term created by the HIGH or LOW operator which has a relocatable term as its operand, the expression cannot be described in the Operand field of the EQU directive.

- If an expression which contains one of the following attribute combinations, an error will result:
  - (a) Expression 1 with ADDRESS attribute - Expression 2 with ADDRESS attribute
  - (b) Expression 1 with ADDRESS attribute Relational operator Expression 2 with ADDRESS attribute provided either of the following conditions ① and ② is met in the expression (a) or (b):
    - ① If label 1 in the expression 1 with ADDRESS attribute and label 2 in the expression 2 with ADDRESS attribute belong to the same segment and if a BR directive for which the number of bytes of the object code cannot be determined instantly is described between the two labels
    - ② If label 1 and label 2 differ in segment and if a BR directive for which the number of bytes of the object code cannot be determined instantly is described between either label and the beginning of the segment to which the label belongs
  - (c) HIGH Absolute expression with ADDRESS attribute
  - (d) LOW Absolute expression with ADDRESS attribute provided the following ③ is met in the expression (c) or (d):
    - ③ If a BR directive for which the number of bytes of the object code cannot be determined instantly is described between the label in the expression with ADDRESS attribute and the beginning of the segment to which the label belongs
- If an error exists in the description format of the operand, the assembler will output an error message but will attempt to store the value of the operand as the value of the name described in the Symbol field to the extent that it can analyze.
- A name defined with the EQU directive cannot be defined again within the same source module.
- A name which has defined a bit value with the EQU directive will have an address and a bit position as its value.

- Table 3-9 shows the bit values that can be described as the operand of the EQU directive. Any of these bit values other than "expression.bit" can be referenced within the same module. "expression.bit1" can be referenced from another module.

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

Operand type	Symbol value	78K/0	78K/I,II	78K/III	78K/VI
A.bit1 (Cf.2)	1.bit1	o	o	o	
X.bit1 (Cf.2)	0.bit1		o	o	
br.bit1 (Cf.1 & 2)	register-name-value1.bit1 (Cf.3)				o
wr.bit2 (Cf.1 & 2)	register-name-value2.bit2 (Cf.3)				
PSW.bit1 (Cf.2)	1FEH.bit1	o	o		
PSWL.bit1(Cf.2)				o	
PSWH.bit1(Cf.2)	1FFH.bit1				
sfr.bit1 (Cf.1 & 2)	0FFxxH.bit1 (Cf.4)	o	o	o	
bsfr.bit1 (Cf.1 & 2)					o
expression. bit1 (Cf.2)	0xxxxH. bit1 (Cf.4)	o	o	o	o

Notes: 1. For details of the bit symbol description, see the user's manual of each device.

2. bit1 = 0 to 7 and bit2 = 0 to 0FH.

3. Register-name-value1:
R0L=0 R0H=1 R1L=2 R1H=3 R2L=4 R2H=5 R3L=6 R3H=7
R4L=8 R4H=9 R5L=0AH R5H=0BH R6L=0CH R6H=0DH
R7L=0EH R7H=0FH
Register-name-vale2:
R0=0 R0=2 R2=4 R3=6 R4=8 R5=0AH R6=0CH R7=0EH

4. 0FFxxH denotes the address of an SFR or bsfr.
0xxxxH denotes the value of an expression.

Application Examples

Example 1

```
            NAME    SAMP1

DATA1       EQU     10H                     ;(1)
DATA2       EQU     20H

ADRS1       EQU     0FE20H                  ;(2)
ADRS2       EQU     0FE21H

            CSEG
             :
            MOV      A, #DATA1              ;(3)
             :
            ADD      A, ADRS1               ;(4)
             :
            MOV      ADRS2 , #DATA1         ;(5)
             :
            END
```

(1) Name "DATA1" has value "10H", symbol attribute "NUMBER", and relocation attribute "ABSOLUTE".

(2) Name "ADRS1" has value "0FE20H", symbol attribute "NUMBER", and relocation attribute "ABSOLUTE".

(3) Name "DATA1" defined in (1) above is described as the operand of the MOV instruction with a value of 10H.

(4) Name "ADRS1" defined in (2) above is described as the operand of the ADD instruction with a value of 0FE20H.

(5) Names "ADRS2" and "DATA1" which have already been defined are described as the operands of the MOV instruction.

If the value "10H" defined as "DATA1" must be changed to 50H, you only need to change 10H to 50H in the directive description (1). Descriptions (3) and (5) need not to be changed.

Example 2

```
             NAME    SAMP2

WORK1        EQU     0FE20H        ;(1)
WORK10       EQU     WORK1.0       ;(2)
WORK11       EQU     WORK1.1       ;(2)

P02          EQU     P0.2          ;(3)
P03          EQU     P0.3          ;(3)

A4           EQU     A.4           ;(4)
X5           EQU     X.5           ;(5)

PSWL5        EQU     PSWL.5        ;(6)
PSWH6        EQU     PSWH.6        ;(7)
             CSEG
              :
             MOV1    CY,WORK10     ;(8)
              :
             MOV1    P02,CY        ;(9)
              :
             OR1     CY,A4         ;(10)
              :
             XOR1    CY,X5         ;(11)
              :
             SET1    PSWL5         ;(12)
              :
             CLR1    PSWH6         ;(13)
              :
             END
```

(1) Name "WORK1" has value "02FE20H", symbol attribute "NUMBER", and relocation attribute "ABSOLUTE".

(2) Bit values "WORK1.0" and "WORK1.1" which are in the operand format "saddr.bit" are assigned names "WORK10" and "WORK11", respectively. Value "0FE20H" has already been defined in (1) for "WORK1" described as the operand of the EQU directive.

(3) Bit values "P0.2" and "P0.3" which are in the operand format "sfr.bit" are assigned names "P02" and "P03", respectively.

(4) Bit value "A.4" which is in the operand format "A.bit" is assigned name "A4".

(5) Bit value "X.5" which is in the operand format "X.bit" is assigned name "X5".

(6) Bit value "PSWL.5" which is in the operand format "PSWL.bit" is assigned name "PSWL5".

(7) Bit value "PSWH.6" which is in the operand format "PSWH.bit" is assigned name "PSWH6".

(8) This description corresponds to "MOV1 CY, saddr.bit".

(9) This description corresponds to "MOV1 sfr.bit, CY".

(10) This description corresponds to "OR1 CY, A.bit".

(11) This description corresponds to "XOR1 CY, X.bit".

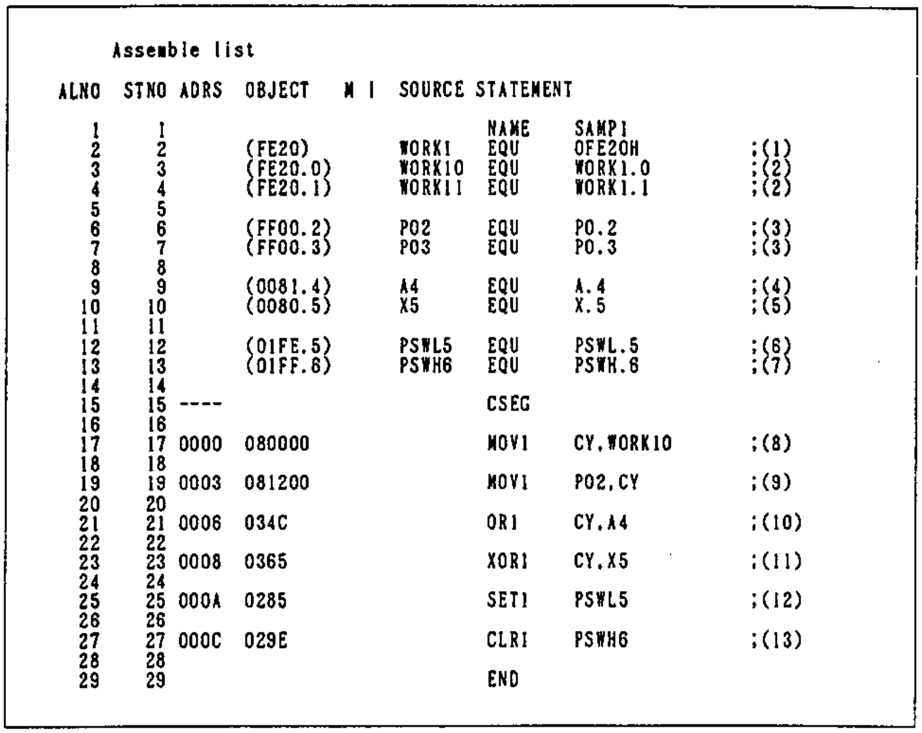
(12) This description corresponds to "SET1 PSWL.bit".

(13) This description corresponds to "CLR1 PSWH.bit".

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

A name which has defined "saddr.bit" can also be referenced as an external definition symbol from another module. (See 3.5 (2), "EXTBIT directive".)

As a result of assembling the source module in Example 2, the following assembly list is generated.



```
     Assemble list

ALNO  STNO ADRS  OBJECT    M I  SOURCE STATEMENT

   1     1                                NAME     SAMP1
   2     2            (FE20)      WORK1   EQU      OFE20H            ;(1)
   3     3            (FE20.0)    WORK10  EQU      WORK1.0           ;(2)
   4     4            (FE20.1)    WORK11  EQU      WORK1.1           ;(2)
   5     5
   6     6            (FF00.2)    P02     EQU      P0.2              ;(3)
   7     7            (FF00.3)    P03     EQU      P0.3              ;(3)
   8     8
   9     9            (0081.4)    A4      EQU      A.4               ;(4)
  10    10            (0080.5)    X5      EQU      X.5               ;(5)
  11    11
  12    12            (01FE.5)    PSWL5   EQU      PSWL.5            ;(6)
  13    13            (01FF.6)    PSWH6   EQU      PSWH.6            ;(7)
  14    14
  15    15 ----                           CSEG
  16    16
  17    17 0000   080000                  MOV1     CY,WORK10         ;(8)
  18    18
  19    19 0003   081200                  MOV1     P02,CY            ;(9)
  20    20
  21    21 0006   034C                    OR1      CY,A4             ;(10)
  22    22
  23    23 0008   0365                    XOR1     CY,X5             ;(11)
  24    24
  25    25 000A   0285                    SET1     PSWL5             ;(12)
  26    26
  27    27 000C   029E                    CLR1     PSWH6             ;(13)
  28    28
  29    29                                END
```

On lines (2) through (7) of the assembly list, the bit address values of the bit values defined as names are indicated in the OBJECT (CODE) column.

(2) EQUD (equate double word)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
name	EQUD	32-bit immediate data or symbol	[;comment]

Function

The EQUD directive defines a name which has the 32-bit value specified in the Operand field.

Use

Define a 32-bit numerical data to be used in the source module as a name with the EQU directive and describe it in the operand of an instruction in place of the numerical data. Numerical data to be frequently used in the source module should be defined as names. By so doing, if you must change a data value in the source module, all you need to do is change the operand value of the name.

Explanation

- A name defined with the EQUD directive can be described in the operand "dword" of an instruction. The defined name has the attribute DNUMBER.
- Only a 32-bit value or a symbol which has already been defined with the EQUD directive can be described in the operand field. No operation can be performed on the operand.
- A name defined with the EQUD directive cannot be defined again within the same source module.
- A name defined with the EQUD directive cannot be either forward or backward referenced.

Application Example

```
L1     EQUD   10000H
L2     EQUD   123456H
L3     EQUD   L1           ;(1)
L4     EQUD   10H
EL1    EQUD   -10H
          ⋮                ;(2)
```

(1) L3 has value "10000H".

(2) An operation is performed in the operand field.

(3) SET (set)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
name	SET	absolute expression	[;comment]

Function

- o The SET directive defines a name which has the value and attributes (symbol attribute and relocation attribute) of the expression specified in the Operand field.
- o The value and attribute of a name defined with the SET directive can be re-defined within the same module.
- o The value and attribute of a name defined with the SET directive are valid until the same name is re-defined.

Use

Define a numerical data (variable) to be used in the source module as a name with the SET directive and describe it in the operand of an instruction in place of the numerical data (variable).

If you wish to change the value of a name in the source module, a different value can be defined for the same name with the SET directive.

Explanation

- o An absolute expression must be described in the Operand field of the SET directive.
- o The SET directive may be described anywhere in a source program. A name which has been defined with the SET directive cannot be forward-referenced.

- o If an error exists in the statement in which a name has been defined with the SET directive, the assembler will output an error message but attempt to store the value of the operand as the value of the name described in the Symbol field to the extent that it can analyze.
- o A symbol (name) defined with the EQU directive cannot be re-defined with the SET directive or vice versa.

Application Example

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

(1) Name "COUNT" has value "10H", symbol attribute "NUMBER", and relocation attribute "ABSOLUTE". The value and attributes are valid until re-definition in (3).

(2) The value "10H" of name "COUNT" will be transferred to register B.

(3) The value of name "COUNT" is changed to 20H.

(4) The value "20H" of name "COUNT" will be transferred to register B.

3.4 Memory Initialization and Area Reservation Directives

Memory initializing directives define constant data to be used in a source program. The value of the defined constant data is generated as an object code.

Area reservation directives reserve memory areas to be used by the source program.

(1) DB (define byte)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	DB	{ (size) [initial value[,...]] }	[;Comment]

Function

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

Use

The DB directive is used when defining an expression or character string to be used in the program.

Explanation

- o If a value in the Operand field is parenthesized, a size is assumed to have been specified. Otherwise, an initial value is assumed.
- o The DB directive cannot be described in a bit segment.

With size specification:

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

With initial value specification:

- o The following two parameters can be specified as initial values:

  ① Expression

  The value of an expression will be secured as an 8-bit data. Therefore, the value of the operand must be in the range of 0 to 0FFH. If the value exceeds 8 bits, the assembler will secure only the low-order 8 bits of the value as initial value data and output an error message.

  ② Character string

  If a character string is described as the operand, an 8-bit ASCII code will be secured for each character in the string.

- o Two or more initial values may be specified within the statement line of the DB directive.
- o As an initial value, an expression which includes a relocatable symbol or external reference symbol may be described.

Application Example

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

← This description is erroneous.

(1) Because "size" is specified, the assembler will initialize each byte area with value "00H".

(2) By this directive, the assembler will initialize a 6-byte area with character string "ABCDEF".

(3) By this directive, the assembler will initialize a 3-byte area with "0AH, 0BH, 0CH".

(4) By this directive, the assembler will initialize a 4-byte area with "00H".

(5) Because the value of expression 'AB'+1 is 4143H (4142H+1) and exceeds the range of 0 to 0FFH, this description will result in an error.

(2) DW (define word)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	DW	(size) [initial value[,...]]	[;Comment]

Function

- o The DW directive tells the assembler to initialize a word area. The number of words to be initialized can be specified as "size".
- o The DW directive also tells the assembler to initialize a memory area in word units (i.e., in units of 2 bytes) with the initial value(s) specified in the Operand field.

Use

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

Explanation

- o If a value in the Operand field is parenthesized, a size is assumed to have been specified. Otherwise, an initial value is assumed.
- o The DW directive cannot be described in a bit segment.

With size specification:

- o If a size is specified in the Operand field, the assembler will initialize an area equivalent to the specified number of words with value "00H".
- o An absolute expression must be described as the size. If the size description is illegal, the assembler will output an error message and will not perform initialization.

With initial value specification:

- o The following two parameters can be specified as initial values:
  ① Constant
  A constant must consist of not more than 16 bits.
  ② Expression
  The value of an expression will be secured as a 16-bit data.
- o No character string can be described as an initial value.
- o The high-order 2 digits of the specified initial value will be stored in the HIGH address and the low-order 2 digits of the value in the LOW address.
- o Two or more initial values may be specified within the statement line of the DW directive.
- o As an initial value, an expression which includes a relocatable symbol or external reference symbol may be described.

Application Example

```
         NAME    SAMPLE
         CSEG
WORK1:   DW      (10)          ;(1)
WORK2:   DW      (128)         ;(1)
         CSEG
         ORG     0H
         DW      MAIN          ;(2)
         DW      SUB1          ;(2)
         CSEG
MAIN:     :
         CSEG
SUB1:     :
DATA:    DW      1234H, 5678H  ;(3)

         END
```

(1) Because "size" is specified, the assembler will initialize each word area with value "00H".

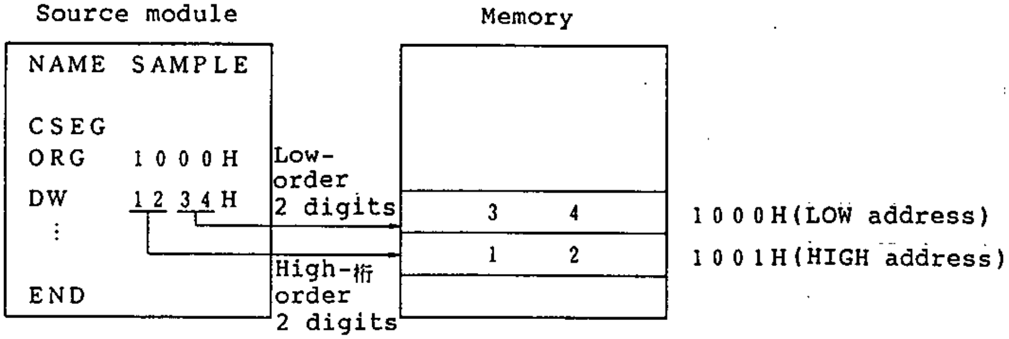
(2) Vector entry addresses are defined with the DW directive.

(3) By this directive, the assembler will initialize a 2-word area with value "34127856".

NOTE 3-1

With a word value, the HIGH (high-order) address of memory is initialized with the high-order 2 digits of the value and the LOW (low-order) address of memory is initialized with the low-order 2 digits of the value.

Example:



Source module

```
NAME  SAMPLE

CSEG
ORG   1000H
DW    1234H
 :

END
```

(3) DS (define storage)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	DS	absolute expression	[;comment]

Function

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

Use

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

Explanation

- o The contents of an area reserved with this directive are unknown.
- o The specified absolute expression will be evaluated with unsigned 16 bits.
- o If the value of the operand is 0, no memory area will be reserved.
- o The DS directive cannot be described within a bit segment.
- o The symbol (label) defined with the DS directive can be referenced only in the backward direction.

- Only the following parameters extended from an absolute expression can be described in the Operand field:
  ① Constant
  ② Expression with a constant expression on which an operation is to be performed (constant expression)
  ③ EQU symbol or SET symbol defined with a constant or constant expression
  ④ Expression 1 with ADDRESS attribute - expression 2 with ADDRESS attribute
  (If label 1 in "expression 1 with ADDRESS attribute and label 2 in "expression 2 with ADDRESS attribute" are relocatable, the labels must have been defined in the same module. However, an error will result in either of the following two cases:
  (a) If label 1 and label 2 belong to the same segment and if a BR instruction for which the number of bytes of the object code cannot be determined instantly is described between the two labels
  (b) If label 1 and label 2 differ in segment and if a BR instruction for which the number of bytes of the object code cannot be determined instantly is described between either label and the beginning of the segment to which the label belong
  ⑤ Any of the expressions ① through ④ above on which an operation is to be performed.
- The following parameters cannot be described in the Operand field:
  ① External reference symbol
  ② Symbol which has defined "expression 1 with ADDRESS attribute - expression 2 with ADDRESS attribute" with the EQU directive
  ③ Location counter ($) if described in either expression 1 or expression 2 in the form of "expression 1 with ADDRESS attribute - expression 2 with ADDRESS attribute"
  ④ Symbol which has defined with the EQU directive an expression with the ADDRESS attribute on which the HIGH or LOW operator is to be operated

Application Example

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

(1) By this directive, the assembler will reserve a 10-byte work area, but the contents of the area are unknown. Label "TABLE1" is assigned to the first address of the area.

(2) By this directive, the assembler will reserve a 1-byte work area.

(3) By this directive, the assembler will reserve a 2-byte work area.

(4) DBIT (define bit)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[name]	DBIT	None	[;comment]

Function

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

Use

The DBIT directive is used to reserve a bit area within a bit segment.

Explanation

- The DBIT directive is described only in a bit segment.
- The contents of a 1-bit area reserved with this directive are unknown.
- If a name is specified in the Symbol field, the name will have an address and a bit position as its value.

Application Example

```
        NAME  SAMPLE
        BSEG
BIT1    DBIT           ;(1)
BIT2    DBIT           ;(1)
BIT3    DBIT           ;(1)
        CSEG
        MOV1  CY,BIT1  ;(2)
          :
        OR1   CY,BIT2  ;(2)
          :
        END
```

(1) By these three DBIT directives, the assembler will reserve three 1-bit areas and define names (BIT1, BIT2, and BIT3) each having an address and a bit position as its value.

(2) This instruction corresponds to "MOV1 CY,saddr.bit" and describes name "BIT1" of the bit area reserved in (1) above as operand "saddr.bit".

(3) This instruction corresponds to "OR1 CY,saddr.bit" and describes name BIT2 as "saddr.bit".

3.5 Linkage Directives

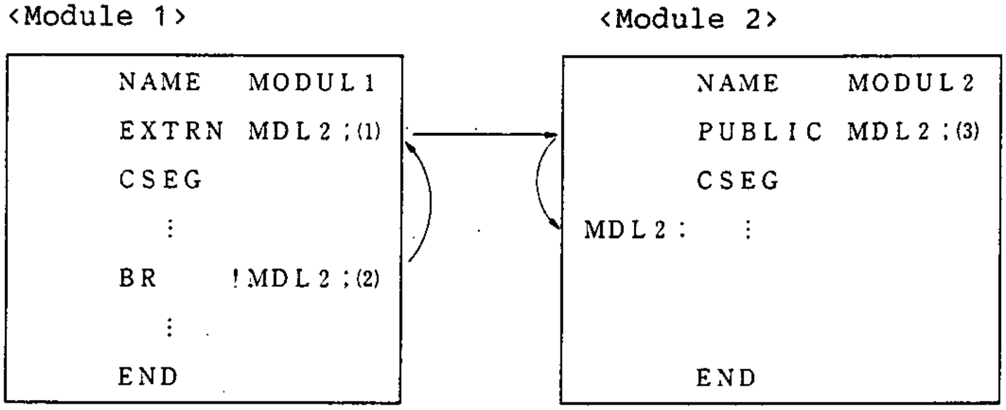
Linkage directives function to make clear the relation between the external definition of a symbol and its external reference. Let's consider a case where a program is created by being divided into two modules: Module 1 and Module 2. In Module 1, if you wish to reference a symbol defined in Module 2, the symbol cannot be used without declaration in each module. For this reason, some sort of signal or indication as "I want to use the symbol" or "You may use the symbol" is required between the two modules.

In Module 1, the <u>external reference declaration</u> of a symbol must be made to indicate that a symbol defined in another module must be referenced. On the other hand, in Module 2, the <u>external definition declaration</u> of a symbol must be made to indicate that the symbol may be referenced in another module.

The symbol can be referenced for the first time when the two external reference and external definition declarations are effectively made.

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

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



<Module 1>

```
      NAME   MODUL1
      EXTRN  MDL2 ;(1)
      CSEG
        :
      BR    !MDL2 ;(2)
        :
      END
```

<Module 2>

```
            NAME    MODUL2
            PUBLIC  MDL2 ;(3)
            CSEG
MDL2 :        :


            END
```

Fig. 3-6. Relationship of Symbols between Two Modules

In Module 1 in Fig. 3-6, the symbol "MDL2" defined in Module 2 is referenced in (2). Therefore, the symbol is declared as an external reference with the EXTRN directive in (1).
In Module 2, the symbol "MDL2" to be referenced from Module 1 is declared as an external definition in (3).
The linker checks whether or not this external reference of the symbol corresponds to the external definition of the symbol.

(1) EXTRN (external)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	EXTRN	symbol name[,...]	[;comment]

Function

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

Use

When referencing a symbol defined in another module, the EXTRN directive must be used to declare the symbol as an external reference.

Explanation

- The EXTRN directive may be described anywhere in a source module. (See Section 2.1, Basic Configuration of Source Program.)
- Up to 20 symbol names may be specified in the Operand field by delimiting each symbol name with a comma (,).
- When referencing a symbol having a bit value, the symbol must be declared as an external reference with the EXTBIT directive.
- The symbol declared with the EXTRN directive must have been declared as PUBLIC in another module.
- No macro name can be described as the operand of the EXTRN directive. (See Chapter 5, Macro for the macro name.)
- A symbol may be declared as EXTRN only once in a module. For the second and subsequent EXTRN declarations for the symbol, the linker will output a warning message.

- A symbol which has been declared with the PUBLIC, EXTBIT, or NAME directive cannot be described as the operand of the EXTRN directive. Conversely, a symbol which has been declared as EXTRN cannot be re-defined or declared with any other directive.
- A symbol which has defined an address in the saddr area with the EXTRN directive can be referenced.
- All symbols declared as EXTRN will be handled as symbols which have the NUMBER attribute.

Application Example

<Module 1>

```
        NAME   SAMP1
        EXTRN  SADR1, SADR2  ;(1)
        CSEG
          :
        MOV    A, SADR1      ;(2)
          :
        MOVW   DE, #SADR2    ;(3)
        MOV    AX, DE
          :
        END
```

<Module 2>

```
               NAME    SAMP2
               PUBLIC  SADR1,SADR2 ;(4)
               DSEG
               ORG     0FE20H
        SADR1: DS      1           ;(5)
        SADR2: DS      2           ;(6)
                 :
               END
```

(1) This directive declares symbols "SADR1" and "SADR2" to be referenced in (2) and (3), respectively, as external references. Two or more symbols may be described in the Operand field.

(2) This instruction references symbol "SADR1".

(3) This instruction references symbol "SADR2".

(4) This directive declares symbols "SADR1" and "SADR2" as external definitions.

(5) This directive defines symbol "SADR1".

(6) This directive defines symbol "SADR2".

(2) EXTBIT (external bit)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	EXTBIT	bit symbol name[,...]	[;comment]

Function

The EXTBIT directive declares to the linker that a bit symbol having a saddr.bit value (bsaddr.bit3 or !addr16.bit3 value with the 78K/VI) in another module is to be referenced in this module.

Use

When referencing a symbol having a bit value (saddr.bit, bsaddr.bit3, or !addr16.bit3) defined in another module, the EXTBIT directive must be used to declare the symbol as an external reference.

Explanation

- o The EXTBIT directive may be described anywhere in a source module.
- o Up to 20 symbol names may be specified in the Operand field by delimiting each symbol name with a comma (,).
- o A symbol declared with the EXTBIT directive must have been declared as PUBLIC in another module.
- o A symbol may be declared as EXTBIT only once in a module. For the second and subsequent EXTBIT declarations for the symbol, the linker will output a warning message.

Application Example

<Module 1>

```
        NAME    SAMP1
        EXTBIT  FLAG1, FLAG2 ;(1)
        CSEG
          :
        MOV1    CY, FLAG1 ;(2)
          :
        OR1     CY, FLAG2 ;(3)
          :
        END
```

<Module 2>

```
        NAME    SAMP2
        PUBLIC  FLAG1, FLAG2 ;(4)
FLAG1   EQU     0FE20H.0     ;(5)
FLAG2   EQU     0FE20H.1     ;(6)
        CSEG
          :


        END
```

(1) This directive declares symbols "FLAG1" and "FLAG2" to be referenced in (2) and (3), respectively, as external references. Two or more symbols may be described in the Operand field.

(2) This instruction references symbol "FLAG1". This description corresponds to "MOV1 CY, saddr.bit".

(3) This instruction references symbol "FLAG2". This description corresponds to "OR1 CY, saddr.bit".

(4) This directive declares symbols "FLAG1" and "FLAG2" as external definitions.

(5) This directive defines symbol "FLAG1".

(6) This directive defines symbol "FLAG2".

(3) PUBLIC (public)

Description Format

Symbol	Mnemonic	Operand	Comment
field	field	field	field
[label:]	PUBLIC	symbol name[,...]	[;comment]

Function

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

Use

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

Explanation

- o The PUBLIC directive must be described in the module header of a source module.
- o Up to 20 symbol names may be specified in the Operand field by delimiting each symbol name with a comma (,).
- o Symbol(s) to be described in the Operand field must have been defined within the same source module.
- o A symbol may be declared as PUBLIC only once in all modules. For the second and subsequent PUBLIC declarations for the symbol, the linker will output a warning message.

- The following symbols cannot be used as the operand of the PUBLIC directive:
  - Name defined with the SET directive
  - Symbol defined with the EXTRN or EXTBIT directive within the same module
  - Name with a bit value other than saddr.bit, bsaddr.bit3, and !addr16.bit3
  - Segment name
  - Module name
  - Macro name

Application Example

Example of program consisting of three modules

<Module 1>

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

A1      EQU     10H
A2      EQU     0FE20H.1

        CSEG
          :
        BR      !B1
          :
        XOR1    CY, C1
          :
        END
```

<Module 2>

```
        NAME    SAMP2
        PUBLIC  B1        ;(2)
        EXTRN   A1
        CSEG
B1:       :
        MOV     C, #LOW(A1)
          :
        END
```

<Module 3>

```
        NAME    SAMP3
        PUBLIC  C1        ;(3)
        EXTBIT  A2
C1      EQU     0FE21H.0

        CSEG
          :
        MOV1    CY, A2
          :
        END
```

(1) This directive declares that symbols "A1" and "A2" are to be referenced from other modules.

(2) This directive declares that symbol "B1" is to be referenced from another module.

(3) This directive declares that symbol "C1" is to be referenced from another module.

3.6 Object Module Name Declaration Directive

The object module name declaration directive NAME gives a module name to an object module to be created by this assembler.

(1) NAME (name)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	NAME	object module name	[;comment]

Function

The NAME directive gives (assigns) the object module name described in the Operand field to an object module to be output by the assembler.

Use

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

Explanation

- The NAME directive may be described anywhere in a source module. For the conventions of module name description, see Subsection 2.2.3 (1), "Symbol field" in Chapter 2.
- No module name can be described as the operand of any directive other than NAME or of any instruction.
- If the NAME directive is omitted, the assembler will assume the primary name of the input source module file as the module name. If two or more module names are specified, the assembler will output a warning message and ignore the second and subsequent module name declarations.
- A module name to be described in the Operand field must not exceed eight characters even if the maximum symbol length is specified as 31 characters with the assembler option (-S).
- The assembler option -CA/-NCA to specify the uppercase/ lowercase for symbol names is valid.

Application Example

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

(1) This directive declares module name "SAMPLE".

3.7 Automatic Branch Instruction Selection Directive

As unconditional branch instructions, which directly describe a branch destination address as their operand, two instructions "BR !addr16" and "BR $addr16" ("BRM $addr16" and "BRS $addr16" with the 78K/VI) are available.

The BR !addr16 instruction or BRM $addr16 are three-byte instructions which allow branching to any address, whereas the BR $addr16 instruction and BRS $addr16 are two-byte instructions which allow branching to an address within the range of -80H to +7FH from the address next to the current location counter value. Therefore, to create a program with high memory utilization efficiency, the 2-byte instruction "BR $addr16" or "BRS $addr16" must be described according to the address range of the branch destination. However, it is quite troublesome to take this address range into account when you describe the branch instruction.

For this reason, there was a need for a directive which directs the assembler to automatically select the two-byte or three-byte branch instruction according to the address range of the branch destination. The BR directive is provided for this purpose.

(1) BR (branch)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	BR	expression	[;comment]

Function

The BR directive causes the assembler to automatically select the 2-byte or 3-byte branch instruction according to the value range of the expression specified in the Operand field and to generate the object code applicable to the selected instruction. This function is referred to as "optimization of branch instructions".

Use

o If the branch destination is within the range of -80H to +7FH from the address next to the current location counter value, you can describe the 2-byte branch instruction "BR $addr16" or "BRS $addr16". With this instruction, required memory space can be reduced by one byte as compared with that when using the 3-byte branch instruction "BR !addr16" or "BRM $addr16". To create a program with high memory utilization efficiency, the 2-byte branch instruction should be used positively. However, each time you describe a branch instruction, it is troublesome for you to take into account the address range of the branch destination. So, use the BR directive when you are not sure of whether or not the the 2-byte branch instruction can be described.

- If it is definite that you can describe the 2-byte or 3-byte branch instruction, describe the applicable branch instruction. In this case, the assembly time can be shortened as compared with that when the BR directive is described.

Explanation

- The BR directive can be described only in a code segment.
- As the operand of the BR directive, describe the branch destination. "$" indicating the current location counter cannot be described at the beginning of an expression to be described in the Operand field.
- For optimization, the following conditions must be satisfied.

  ① The number of labels or forward-referenced symbols in the expression is 1 or less.

  ② An EQU symbol with the ADDRESS attribute has not been described.

  ③ A symbol which has defined "expression 1 with ADDRESS attribute - expression 2 with ADDRESS attribute" with the EQU directive has not been described.

  ④ An expression with ADDRESS attribute on which the HIGH or LOW operator is to be operated has not been described.

  If these conditions are not met, the 3-byte BR instruction will be selected.
- The optimization conditions of the BR directive are shown in Table 3-10 below.

Table 3-10. Optimization Conditions of BR Directive

Jump destination condition	Jump source condition	Absolute segment		Relocatable segment	
	Reference direction	Backward	Forward	Backward	Forward
Numeric value		Optimize	Optimize	3-byte BR	3-byte BR
Name (symbol attribute: NUMBER)		Optimize	Optimize	3-byte BR	3-byte BR
Label	Same segment	Optimize	Optimize	Optimize	Optimize
	Same named segment	Optimize	Optimize	Optimize	Optimize
	Another segment (same type)	Optimize	Optimize	3-byte BR	3-byte BR
	Another segment (another type)	3-byte BR	3-byte BR	3-byte BR	3-byte BR
External reference name		3-byte BR	3-byte BR	3-byte BR	3-byte BR
Location counter ($)		Optimize	-	Optimize	-

NOTE: o "-" in the table indicates that the combination is prohibited.

o "Backward" reference denotes the reference of a symbol which has already been defined in the source module.

o "Forward" reference denotes the reference of a symbol which is to be defined in a subsequent line.

Application Example

```
        NAME  SAMPLE
        CSEG
         :
L1:     MOV  A, #10H
         :                       Relocatable
        BR   L1        ;(1)      segment
        BR   L2        ;(2)
         :
S1      CSEG  AT  1000H
L2:      :                       Absolute
        BR   L2        ;(3)      segment
        BR   L1         (4)
         :
        END
```

(1) This BR directive will be optimized.
If displacement between the line (1) and the "L1:" label definition is within -80H, the object code of a 2-byte branch instruction will be generated.

(2) This BR directive will be substituted with a 3-byte branch instruction, because it branches to a label in another segment.

(3) This BR directive will be optimized.
If displacement between the line (3) and the "L2:" label definition is within -80H, the object code of a 2-byte branch instruction will generated.

(4) Because the relocation attribute of "L1" described as the operand of this BR directive is a relocatable term, the object code of a 3-byte branch instruction will be generated.

3.8 General-purpose Register Selection Directive (applicable to 78K/III only)

With the general-purpose registers of the 78K/III, correspondence of their function names to their absolute names is different depending on the value of the Register Set Select (RSS) flag in the PSW. (See Table 3-11 below.)

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

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

Table 3-11. Absolute Names and Function Names of General-purpose Registers

Absolute name	Function name	
	RSS=0	RSS=1
R0	X	
R1	A	
R2	C	
R3	B	
R4		X
R5		A
R6		C
R7		B
R8	VP_L	VP_L
R9	VP_H	VP_H
R10	UP_L	UP_L
R11	UP_H	UP_H
R12	E	E
R13	D	D
R14	L	L
R15	H	H

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

NOTE: A blank column in the table indicates that by describing the absolute name, the corresponding register can be accessed.

(1) RSS (register set select)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	RSS	absolute value with evaluated value 0 or 1	[;comment]

Function

- o The RSS directive is a directive dedicated to the 78K/III.
- o The RSS directive causes the assembler to generate object codes by substituting the general-purpose registers of the function names described in the source program with those of the corresponding absolute names, based on the value of the Register Set Select (RSS) flag specified in the Operand field.

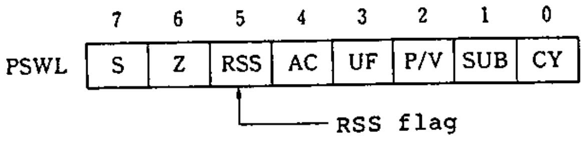
  See Table 3-11 for the function names and absolute names of the general-purpose registers.

Use

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

Explanation

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



	7	6	5	4	3	2	1	0
PSWL	S	Z	RSS	AC	UF	P/V	SUB	CY

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

NOTE 3-2

During the branch destination processing by the context switching function of the 78K/III, the value of the RSS flag must be 0. (This is because of that the PC and PSW values required for restoration from the branch destination processing have been saved to the general-purpose registers R4 to R7.) For this reason, when describing a branch destination processing routine by the context switching function by using the function names of general-purpose registers, the value of the RSS flag must not be set to "1" in the routine.

Application Examples

Example 1

```
        NAME    SAMP1
        CSEG
SUB1:   MOV     B, A          ;(1)
        MOV     A, C          ;(2)
        RET
        CSEG
SUB2:   RSS     1             ;(3)
        SET1    PSWL.5        ;(4)
        MOV     B, A          ;(5)
        RET
SUB3:   RSS     0             ;(6)
        CLR1    PSWL.5        ;(7)
        MOV     B, A          ;(8)
        RET
SUB4:   RSS     1             ;(9)
        SWRS                  ;(10)
        MOV     B, A          ;(11)
        RET

        END
```

(1) The default value of RSS in the assembler is "0". Because the RSS directive is omitted, this description corresponds to "MOV R3,R1".

(2) This description corresponds to "MOV A,R2".

(3) The RSS directive must be described immediately before (or after) the instruction which sets the RSS flag in (4).

(5) This description corresponds to "MOV R7,R5".

(6) The RSS directive must be described immediately before (or after) the instruction which resets the RSS flag in (7).

(8) This description corresponds to "MOV R3,R1".

(9) The RSS directive must be described immediately before (or after) the instruction which switches the RSS flag in (10).

(11) This description corresponds to "MOV R7,R5".

See the following assembly list for the object codes to be generated from assembly of the source program in Example 1.

```
     Assemble list

ALNO  STNO ADRS  OBJECT   M I  SOURCE STATEMENT

  1     1                              NAME    SAMP1
  2     2 ----                         CSEG
  3     3 0000  2431           SUB1:   MOV     B,A              ;(1)
  4     4 0002  D2                     MOV     A,C              ;(2)
  5     5 0003  56                     RET
  6     6 ----                         CSEG
  7     7 0004                 SUB2:   RSS     1                ;(3)
  8     8 0004  0285                   SET1    PSWL.5           ;(4)
  9     9 0006  2475                   MOV     B,A              ;(5)
 10    10 0008  56                     RET
 11    11 0009                 SUB3:   RSS     0                ;(6)
 12    12 0009  0295                   CLR1    PSWL.5           ;(7)
 13    13 000B  2431                   MOV     B.A              ;(8)
 14    14 000D  56                     RET
 15    15 000E                 SUB4:   RSS     1                ;(9)
 16    16 000E  43                     SWRS                     ;(10)
 17    17 000F  2475                   MOV     B,A              ;(11)
 18    18 0011  56                     RET
 19    19
 20    20                              END
```

Example 2

```
        NAME  SAMP2
        CSEG
SUB5:   SET1  PSWL.5        ;(1)
        MOV   B,A           ;(2)
        RET
SUB6:   CLR1  PSWL.5        ;(3)
        MOV   B,A           ;(4)
        RET

        END
```

(1) The RSS flag is set. However, the RSS directive has not been described immediately before or after the SET1 instruction.

(2) This description corresponds to "MOV R3,R1". The object code expected for "MOV R7,R5" will not be generated.

(3) The RSS flag is reset. However, the RSS directive has not been described immediately before or after the CLR1 instruction.

(4) This description corresponds to "MOV R3,R1".

See the following assembly list for the object codes to be generated from assembly of the source program in Example 2.

3.9 Macro Directives

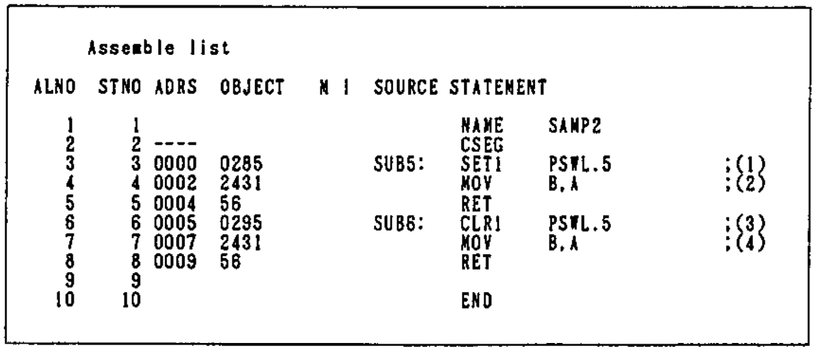
When you describe a source program, it is troublesome for you to describe a series of frequently used instruction groups over and over again, and this may cause an increase in the number of description or coding errors.

By using the macro function with macro directives, the need to repeatedly describe the same group of instructions can be eliminated, thereby increasing coding efficiency of the program.

The basic function of a macro is the substitution of a series of statements with a name. For details of the macro function, see Chapter 5, Macros.

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

In this section, each of these directives is detailed.



```
     Assemble list

ALNO  STNO ADRS  OBJECT   M I  SOURCE STATEMENT

   1     1                            NAME    SAMP2
   2     2 ----                       CSEG
   3     3 0000  0285          SUB5:  SET1    PSWL.5            ;(1)
   4     4 0002  2431                 MOV     B,A               ;(2)
   5     5 0004  56                   RET
   6     6 0005  0295          SUB6:  CLR1    PSWL.5            ;(3)
   7     7 0007  2431                 MOV     B,A               ;(4)
   8     8 0009  56                   RET
   9     9
  10    10                            END
```

(1) MACRO (macro)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
macro name	MACRO	[formal parameter[,...]]	[;comment]
	:		
	macro body		
	:		
	ENDM		[;comment]

Function

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

Use

Define a series of frequently used statements in the source program with a macro name. After the macrodefinition, you only need to describe the defined macro name (for macro reference) and the macro body corresponding to the macro name will be expanded.

Explanation

- The MACRO directive must be paired with the ENDM directive.
- For the macro name to be described in the Symbol field, see the conventions of symbol description in Subsection 2.2.3 (1), "Symbol field" in Chapter 2.
- To reference a macro, describe the defined macro name in the Mnemonic field. (See Application Example.)
- For the formal parameter(s) to be described in the Operand field, the same rules as the conventions of symbol description will apply.

- o Up to 16 formal parameters can be described per MACRO directive.
- o Formal parameters are valid only within the macro body.
- o An error will result if any reserved word is described as a formal parameter. However, if a user-defined symbol is described, its recognition as a formal parameter will take precedence.
- o The number of formal parameters must be the same as the number of actual parameters.
- o A name or label defined within the macro body if declared with the LOCAL directive becomes effective with respect to one-time macroexpansion.
- o Nesting of macros (i.e., to refer to other macros within the macro body) is allowed up to eight levels including REPT and IRP directives.
- o The number of macros that can be defined within a single source module is not specifically limited. In other words, macros may be defined as long as the memory space is available.
- o Formal parameter definition lines, reference lines, and symbol names will not be output to a cross-reference list.

Application Example

```
          NAME      SAMP

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

Macro body

(1) A macro is defined by specifying macro name "ADMAC" and two formal parameters "PARA1" and "PARA2".

(2) This directive indicates the end of the macrodefinition.

(3) Macro "ADMAC" is referenced.

(2) LOCAL (local)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	LOCAL	symbol name[,...]	[;comment]

Function

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

Use

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

Explanation

- A symbol declared as LOCAL will be substituted with a symbol "??RAn (where n=0000 to FFFF) at each macroexpansion. The symbol "??RAn" after the macro replacement will be handled the same as a global symbol and will be stored in the symbol table and thus can be referenced under the symbol name ??RAn".
- If a symbol is defined within a macro body and the macro is referenced more than once, it means that the symbol would be defined more than once in the source module. For this reason, it is necessary to declare that the symbol is a local symbol which is valid only within the macro.
- The LOCAL directive can be used only within a macro-definition.

- o The LOCAL directive must be described before using the symbol specified in the Symbol field. (In other words, the LOCAL directive must be described at the beginning of the macro body.)
- o Symbol names to be defined with the LOCAL directive within a single source module must be all different. (In other words, the same name cannot be used for local symbols to be used in each macro.)
- o The number of symbol names that can be specified in the Operand field is not limited as long as they are all within a line. However, the number of symbols within a macro body is limited to 64. If more than 65 local symbols are declared, an error message will be output and the macro-definition will be stored as an empty macro body. Nothing will be expanded even if the macro is called.
- o Macros defined with the LOCAL directive cannot be nested.
- o Symbols defined with the LOCAL directive cannot be called (referenced) from outside the macro.
- o No reserved word can be described as a symbol name in the Operand field. However, if a user-defined symbol is described, its recognition as a local symbol will take precedence.
- o A symbol declared as the operand of the LOCAL directive will not be output to a cross-reference list.
- o The statement line of the LOCAL directive will not be output at the time of the macroexpansion.

Application Example

<Source program>

```
            NAME    SANPLE
MAC1        MACRO
            LOCAL   LLAB          ;(1)
LLAB          :                         Macrodefinition
            BR      $LLAB         ;(2)
            ENDM
              :
REF1:       MAC1                  ;(3)
              :
            BR      !LLAB         ;(4) ←This description is
                                         erroneous.
              :
REF2:       MAC1                  ;(5)
              :
            END
```

(1) This directive defines symbol name "LLAB as a local symbol.

(2) This instruction references local symbol "LLAB" within macro MAC1.

(3) This directive references macro MAC1.

(4) Because local symbol "LLAB" is referenced outside the definition of macro MAC1, this description causes an error.

(5) This directive references macro MAC1.

If the source program in the above example is assembled, macroexpansion (replacement of a macrocall by the body itself) occurs as shown below.

<Assembly list>

```
          NAME     SAMPLE
MAC1      MACRO
          LOCAL    LLAB
LLAB:       :                      Macrodefinition
          BR       $LLAB
          ENDM
            :
REF1:     MAC1
          LOCAL    LLAB
LLAB:       :                      Macroexpansion
          BR       $LLAB
            :
          BR       !LLAB          ←This description is
                                   erroneous.
            :
REF2:     MAC1
          LOCAL    LLAB
LLAB:       :                      Macroexpansion
          BR       $LLAB
            :
          END
```

(3) REPT (repeat)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	REPT	absolute expression	[;comment]
	:		
	ENDM		[,comment]

Function

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

Use

If a series of statements is to be described repeatedly in a source program, use the REPT-ENDM block.

Explanation

- o An error will result if the REPT directive is not paired with the ENDM directive.
- o In the REPT-ENDM block, macro references, REPT and IRP ENDM directives can be nested up to 8 levels.
- o If the EXITM directive appears in the REPT-ENDM block, subsequent expansion of the REPT-ENDM block by the assembler will be terminated.
- o Assembly control instructions may be described in the REPT-ENDM block.
- o The absolute expression described in the Operand field will be evaluated with unsigned 16 bits. If the value of the expression is 0, nothing will be expanded.

Application Examples

Example 1 <Source program>

```
        NAME    SAMP1

        CSEG
         :
        REPT    3          ;(1)
        INC     B
        DEC     C
        ENDM               ;(2)
         :
        END
```

REPT-ENDM block

(1) This directive instructs the assembler to expand the REPT-ENDM block three consecutive times.

(2) This directive indicates the end of the REPT-ENDM block.

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

<Assembly list>

```
        NAME    SANP1

        CSEG
         :
        INC       B
        DEC       C
        INC       B
        DEC       C
        INC       B
        DEC       C
         :
        END
```

You can see that the REPT-ENDM block defined by statements (1) and (2) has been expanded three times. On the assembly list, the definition statements (1) and (2) by the REPT directive in the source module will not be displayed.

Example 2 <Source program>

```
        NAME    SAMP2

        CSEG
          :
        REPT    3           ;(1)
        INC     B
        REPT    2           ;(2)
        DEC     C
        ENDM
        ENDM
          :
        END
```

(1) This directive instructs the assembler to expand the REPT-ENDM block three consecutive times.

(2) This directive instructs the expansion of the REPT-ENDM block again within the REPT-ENDM block. Within the REPT-ENDM block, nesting of macro references, REPT and IRP is allowed up to 8 levels.

(4) IRP (indefinite repeat)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	IRP	formal parameter, <actual parameter[,...]>	[;comment]
	:		
	ENDM		[;comment]

Function

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

Use

If a series of statements, only part of which becomes variables is to be described repeatedly in a source program, use the IRP-ENDM block.

Explanation

- o The IRP directive must be paired with the ENDM directive.
- o Up to 16 actual parameters may be described in the Operand field.
- o In the IRP-ENDM block, macro references, REPT and IRP ENDM directives can be nested up to 8 levels.
- o If the EXITM directive appears in the IRP-ENDM block, subsequent expansion of the IRP-ENDM block by the assembler will be terminated.
- o No macro can be defined in the IRP-ENDM block.

o Assembly control instructions may be described in the REPT-ENDM block.

Application Example

<Source program>

```
        NAME    SAMP1

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

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

(2) This directive indicates the end of the IRP-ENDM block.

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

<Assembly list>

```
        NAME    SAMP1

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

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

(3) In this instruction, PARA has been replaced with 0AH.

(4) In this instruction, PARA has been replaced with 0BH.

(5) In this instruction, PARA has been replaced with 0CH.

(5) EXITM (exit from macro)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	EXITM	None	[;comment]

Function

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

Use

o This function is mainly used when a conditional assembly function (see Section 4.7, Conditional Assembly Control Instructions) is used in the macro body defined with the MACRO directive.

o If conditional assembly functions are used in combination within the macro body, part of the source program which must not be assembled is likely to be assembled unless control is returned from the macro by force with the EXITM directive. In such a case, the EXITM directive must be used.

Explanation

o If the EXITM directive is described in a macro body, instructions up to the ENDM directive will be stored as the macro body.

o The EXITM directive indicates the end of a macro only during the macroexpansion.

o If something is described in the Operand field of the ENDM directive, the assembler will output an error message but execute the EXITM processing.

- o If the EXITM directive appears in a macro body, the assembler will return by force the nesting level of IF/_IF/ ELSE/ELSEIF/_ELSEIF/ENDIF blocks to the level when the assembler entered the macro body.
- o If the EXITM directive appears in an Include file resulting from expanding the INCLUDE control instruction described in a macro body, the assembler will accept the EXITM directive as valid and terminate the macroexpansion at that level.

Application Example

- o In the example here, conditional assembly control instructions are used. See Section 4.7, Chapter 4 for the conditional assembly control instructions.
- o See Chapter 5, Macros for the macro body and macro-expansion.

<Source program>

```
            NAME      SAMP1
MAC1        MACRO                  ;(1)
            NOT1      A.1                     Macro body
$           IF (SW1)               ;(2)  ┐
            BT        A.1, $L1           │ IF block
            EXITM                  ;(3)  ┘
$           ELSE                   ;(4)  ┐
            MOV1      CY, A.1            │ ELSE block
            MOV       A, #0              │
$           ENDIF                  ;(5)  ┘
$           IF (SW2)               ;(6)  ┐
            BR        [HL]               ┘ IF block
$           ELSE                   ;(7)  ┐
            BR        [DE]               ┘ ELSE block
$           ENDIF                  ;(8)
            ENDM                   ;(9)
            CSEG
$           SET (SW1)              ;(10)
            MAC1                   ;(11) ←──Macro reference
            NOP
L1:         NOP
            END
```

(1) The macro "MAC1" uses conditional assembly functions (2) and (4) through (8) within the macro body.

(2) This instruction defines an IF block for conditional assembly. If switch name "SW1" is true (0FFH), the IF block will be assembled.

(3) This directive terminates by force the expansion of the macro body in (4) and thereafter.
If this EXITM directive is omitted, the assembler will proceed to the assembly process in (6) and thereafter when the macro is expanded.

(4) This instruction defines an ELSE block for conditional assembly. If switch name "SW1" is false (00H), the ELSE block will be assembled.

(5) This instruction indicates the end of the conditional assembly.

(6) This instruction defines another IF block for conditional assembly. If switch name "SW2" is true (0FFH), the IF block following this will be assembled.

(7) This instruction defines another ELSE block for conditional assembly. If switch name "SW2" is false (00H), the ELSE block will be assembled.

(8) This instruction indicates the end of the conditional assembly processes in (6) and (7).

(9) This directive indicates the end of the macro body.

(10) This SET control instruction gives true value (0FFH) to switch name "SW1" and sets the condition of the conditional assembly.

(11) This instruction references macro "MAC1".

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

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

Macro-expanded part

By the macro reference in (11), the macro body of macro "MAC1" has been expanded. Because true value (0FFH) is set in switch name "SW1" in (10), the first IF block in the macro body is assembled. Because the EXITM directive is described at the end of the IF block, the subsequent macroexpansion is not executed.

(6) ENDM (end macro)

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
None	ENDM	None	[;comment]

Function

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

Use

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

Explanation

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

Application Examples

Example 1 <MACRO-ENDM>

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

Example 2 <REPT-ENDM>

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

Example 3 <IRP-ENDM>

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

3.10 Assembly Termination Directive

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

The assembler processes a series of statements up to the assembly termination directive as a source module. Therefore, if an REPT-ENDM block or IRP-ENDM block exits before the END directive, the REPT-ENDM block or IRP-ENDM block will become invalid.

(1) END (end)

Description Format

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

Function

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

Use

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

Explanation

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

Application Example

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

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

CHAPTER 4. CONTROL INSTRUCTIONS

4.1 Overview of Control Instructions

Control instructions are described in a source program to provide particular instructions on the assembler operation. These instructions are not subject to object code generation.

Control instructions are available in the following six types:

Table 4-1. List of Control Instructions

No.	Type of control instruction	Control instruction
1	Instruction to specify the processor type for the target device subject to assembly	PROCESSOR
2	Instructions to control debug information output	DEBUG, NODEBUG
3	Instructions to control cross-reference list output	XREF, NOXREF
4	Instruction to control INCLUDE file	INCLUDE
5	Instructions to control assembly list	EJECT, LIST, NOLIST, GEN, NOGEN, COND, NOCOND, TITLE, SUBTITLE

6	Instructions to control conditional assembly	SET, RESET
		IF, _IF, ELSEIF, _ELSEIF, ELSE, ENDIF

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

Of the control instructions listed in Table 4-1, the following instructions has the same functions as assembler options which can be specified in the start-up command line of the assembler:

Control instruction	Assembler option
PROCESSOR	-C
DEBUG/NODEBUG	-G/-NG
XREF/NOXREF	-KX/-NKX
TITLE	-LH

For the method of specifying assembler options in the start-up command line, see Subsection 4.3.1, "Starting up the assembler", Chapter 4 in the RA78K Series Assembler Package User's Manual for Operation.

4.2 Processor Type Specification Control Instruction

The processor type control instruction (PROCESSOR) is used to specify in a source module file the processor type for the target device (target chip) subject to assembly.

(1) PROCESSOR (processor)

Description Format

```
[△]$[△]PROCESSOR[△] ([△]processor type)[△])
[△]$[△]PC[△]([△]processor type[△])           ; Abbreviated
                                               format
```

Function

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

Use

- o The processor type for the target device subject to assembly must always be specified in either the header section of a source module file or the start-up command line of the assembler.
- o If you omit the processor type specification for the target device subject to assembly in each source module file, you must specify the processor type at each assembly operation. Therefore, by specifying the processor type for the target device subject to assembly in the source module file, you may save your trouble when starting up the assembler.

Explanation

- o Specify one of the processor types for the target device subject to assembly listed in Table 4-2 below.
- o The PROCESSOR control instruction can be described only in the header section of a source module file. If the instruction is described elsewhere, the assembler will be aborted.
- o If the specified processor type differs from the actual target device subject to assembly, the assembler will be aborted.

Table 4-2. Processor Types for Target Devices

Series	Target Device	Processor type
78K/0	uPD78012	012
	uPD78014, uPD78P014	014
78K/I	uPD78112, uPD78P112	112
	uPD78134, uPD78P134	134
	uPD78136	136
78K/II	uPD78210	210
	uPD78212	212
	uPD78213	213
	uPD78214, uPD78P214	214
	uPD78220	220
	uPD78224, uPD78P224	224
	uPD78233	233
	uPD78234	234
78K/III	uPD78310	310
	uPD78312, uPD78P312	312
	uPD78310A	310A
	uPD78312A	312A
	uPD78320	320
	uPD78322	322
	uPD78330	330
	uPD78334	334
78K/VI	uPD78600	600
	uPD78602	602

o Only one PROCESSOR control instruction can be specified in the module header.

o The processor type for the target device subject to assembly may also be specified with the assembler option -C in the start-up command line of the assembler. If the specified processor type differs between the source module file and the start-up command line, the assembler will output a warning message and give precedence to the processor type specification in the start-up command line.

o If the processor type is not specified in either the source module file or the start-up command line, the assembler will be aborted.

Application Example

```
$       PROCESSOR(310)
$       DEBUG
$       XREF

        NAME        TEST

        CSEG

          ⋮
```

4.3 Debug Information Output Control Instructions

Debug information output control instructions (DEBUG and NODEBUG) are used to specify in a source module file the output or non-output of debugging information to an object module file created from the source module file.

(1) DEBUG/NODEBUG (debug/nodebug)

Description Format

```
[△]$[△]DEBUG
[△]$[△]DG                                  ; Abbreviated format
[△]$[△]NODEBUG
[△]$[△]NODG                                ; Abbreviated format
```

Function

- o The DEBUG control instruction indicates to the assembler the output of local symbol information to an object module file.
- o The NODEBUG control instruction indicates to the assembler the non-output of local symbol information to an object module file.
- o The local symbol information refers to information on symbols other than module names and those declared with PUBLIC, EXTRN, and EXTBIT directives.

Use

- o Specify the DEBUG control instruction when a program is to be debugged.
- o If you must specify the output or non-output of debug information at each assembly operation, you may save your time and labor by specifying the DEBUG or NODEBUG control instruction in the source module file.

Explanation

- o The DEBUG or NODEBUG control instruction can be described only in the header section of a source module file.
- o If two or more of these control instructions are specified at the same time, the last specified control instruction will take precedence over the others.

- o The output or non-output of debug information to an object module file may also be specified with the assembler option -G or -NG in the start-up command line of the assembler.
- o If the debug information output specification differs between the source module file and the start-up command line, the assembler will give precedence to the specification by the start-up command line.
- o The assembler will perform a syntax check on the DEBUG or NODEBUG control instruction even when the assembler option -NO (non-output of object module file) has been specified in the start-up command line.

Application Example

See 4.2 (1), PROCESSOR control instruction for the application example of the DEBUG control instruction.

4.4 Cross-reference List Output Control Instructions

Cross-reference list output control instructions (XREF and NOXREF) are used in a source module file to specify the output or non-output of a cross-reference list to an assembly list file to be output by the assembler.

(1) XREF/NOXREF (xref/noxref)

Description Format

```
[△]$[△]XREF
[△]$[△]XR                         ; Abbreviated format
[△]$[△]NOXREF
[△]$[△]NOXR                       ; Abbreviated format
```

Function

- o The XREF control instruction indicates to the assembler the output of a cross-reference list to an assembly list file to be output by the assembler.
- o The NOXREF control instruction indicates to the assembler the non-output of a cross-reference list.

Use

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

Explanation

- o The XREF or NOXREF control instruction can be described only in the header section of a source module file.
- o If two or more of these control instructions are specified at the same time, the last specified control instruction will take precedence over the others.

- o The output or non-output of a cross-reference list to an assembly list file may also be specified with the assembler option -KX or -NKX in the start-up command line of the assembler.
- o If the cross-reference list output specification differs between the source module file and the start-up command line, the assembler will give precedence to the specification by the start-up command line.
- o The assembler will perform a syntax check on the XREF or NOXREF control instruction even when the assembler option -NP (non-output of assembly list file) has been specified in the start-up command line.

Application Example

See 4.2 (1), PROCESSOR control instruction for the application example of the XREF control instruction.

4.5 INCLUDE control instruction

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

By making the most of this control instruction, you may save your time and labor in describing a source program.

(1) INCLUDE (include)

Description Format

```
[△]$[△]INCLUDE[△]([△]filename[△])
[△]$[△]IC[△]([△]filename[△])          ; Abbreviated format
```

Function

The INCLUDE control instruction inserts the contents of the file specified by "filename" into the source program for assembly.

Use

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

Explanation

- o The pathname or drive name of an INCLUDE file may be specified with the assembler option -I.
- o Include file read paths are searched in the following sequence:

  (a) When an Include file is specified without pathname specification

  1 Path in which the source file exists

  2 Path specified by the assembler option -I

  3 Path specified by environment variable INC78Kn (where n = 0, 1, 2, 3, or 6 corresponding to each series number)

(b) When an Include file is specified with a pathname
If the Include file is specified with a drive name or a pathname which begins with "¥", the path specified with the Include file will be prefixed to the Include filename. If the Include file is specified with a relative path (which does not begin with "¥"), a pathname will be prefixed to the Include filename in the order described in (a) above.

o Nesting of INCLUDE files is allowed up to one level. In other words, the nesting level display of Include files in the assembly list is up to 2 (i.e., I1 and I2). (The term "nesting" used here refers to the specification of one or more other INCLUDE files in an INCLUDE file.)

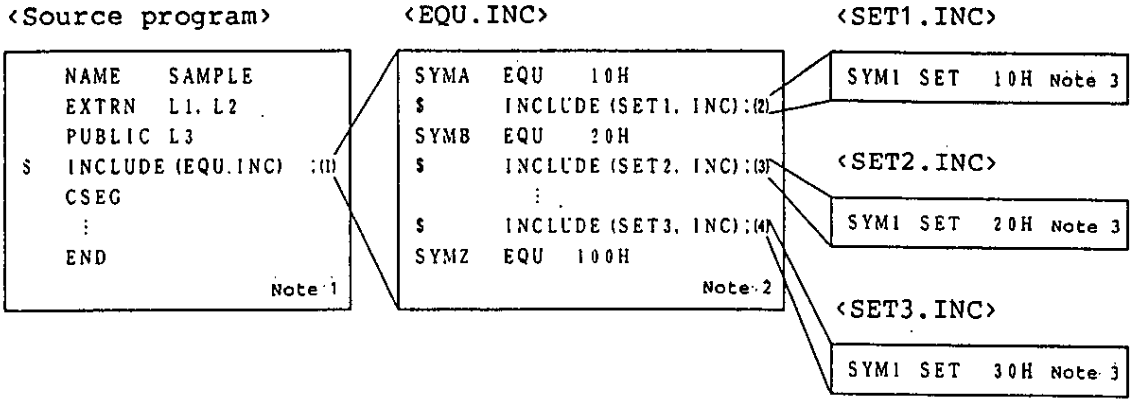
o The END directive need not be described in an INCLUDE file.

o If the specified Include file cannot be opened, the assembler will be aborted.

o An Include file must be closed with an IF or _IF control instruction being properly paired with an ENDIF control instruction within the Include file. If the IF level at the entry of the Include file expansion does not correspond with the IF level immediately after the Include file expansion, the assembler will output an error message and force the IF level to return to that level at the entry of the Include file expansion.

o When defining a macro in an Include file, the macrodefinition must be closed in the Include file. If an ENDM directive appears unexpectedly (without the corresponding MACRO directive) in the Include file, an error message will be output and the ENDM directive will be ignored. If an ENDM directive is missing for the MACRO directive described in the Include file, the assembler will output an error message but will process the macrodefinition by assuming that the corresponding ENDM directive has been described.

Application Example



<Source program>

```
    NAME    SAMPLE
    EXTRN   L1, L2
    PUBLIC  L3
$   INCLUDE (EQU.INC)  ;(1)
    CSEG
     :
    END
                    Note 1
```

<EQU.INC>

```
SYMA  EQU     10H
$     INCLUDE (SET1, INC);(2)
SYMB  EQU     20H
$     INCLUDE (SET2, INC);(3)
        :
$     INCLUDE (SET3, INC);(4)
SYMZ  EQU     100H
                         Note 2
```

<SET1.INC>

```
SYM1 SET  10H Note 3
```

<SET2.INC>

```
SYM1 SET  20H Note 3
```

<SET3.INC>

```
SYM1 SET  30H Note 3
```

Notes:
1. Two or more $IC control instructions can be specified in the source file. The same Include file may also be specified two or more times.
2. Two or more $IC control instructions may be specified for Include file "EQU.INC".
3. No $IC control instruction can be specified in any of the Include files "SET1.INC", "SET2.INC", and "SET3.INC".

(1) This control instruction specifies "EQU.INC" as the INCLUDE file. When this source program is assembled, the contents of the INCLUDE file will be expanded as follows:

```
        NAME    SAMPLE
        EXTRN   L1, L2
        PUBLIC  L3
$       INCLUDE (EQU.INC)   ;(1)

SYMA    EQU     10H
$       INCLUDE (SET1.INC)  ;(2)

SYM1    SET     10H

SYMB    EQU     20H
$       INCLUDE (SET2.INC);(3)

SYM1    SET     20H


$       INCLUDE (SET3.INC);(4)

SYM1    SET     30H

SYMZ    EQU     100H

        CSEG
        END
```

←The contents of INCLUDE file "EQU.INC" have been expanded.

←The contents of INCLUDE file "SET1.INC" have been expanded.

←The contents of INCLUDE file "SET2.INC" have been expanded.

←The contents of INCLUDE file "SET3.INC" have been expanded.

4.6 Assembly List Control Instructions

Assembly list control instructions are used in a source module file to control the output format of an assembly list such as page ejection, suppression of list output, title output, and subtitle output.

These control instructions include:

- o EJECT
- o LIST and NOLIST
- o GEN and NOGEN
- o COND and NOCOND
- o TITLE
- o SUBTITLE

(1) EJECT (eject)

Description Format

```
[△]$[△]EJECT
[△]$[△]EJ                              ; Abbreviated format
```

Function

The EJECT control instruction causes the assembler to execute page ejection (formfeed) of an assembly list.

Use

Describe the EJECT control instruction in a line of the source module at which the page ejection of the assembly list is required.

Explanation

- o Page ejection of the assembly list takes place after the image (i.e., $ EJECT) of the EJECT control instruction itself has been printed.
- o If the assembler option "-NP" or "-LL0" is specified in the start-up command line or if the assembly list output is disabled by another control instruction, the EJECT control instruction will become invalid. (See the "RA78K Series Assembler Package User's Manual for Operation" for the assembler options -NP and -LL.)
- o If an illegal description follows the EJECT control instruction, the assembler will output an error message.

Application Example

<Source module>

```
        :
      MOV      [DE+], A
      BR       $$
$     EJECT                  ;(1)
      CSEG
        :
      END
```

(1) When page ejection is executed with the EJECT control instruction, the output assembly list will look like this.

```
        :
      MOV      [DE+], A
      BR       $$
$     EJECT
------------------------------------------ ←Page ejection
      CSEG
        :
      END
```

(2) LIST/NOLIST (list/no list)

Description Format

```
[△]$[△]LIST
[△]$[△]LI                               ; Abbreviated format
[△]$[△]NOLIST
[△]$[△]NOLI                             ; Abbreviated format
```

Function

- o The LIST control instruction indicates to the assembler the line at which assembly list output must start.
- o The NOLIST control instruction indicates to the assembler the line at which assembly list output must be suppressed. All source statements described after the NOLIST control instruction specification until the LIST control instruction appears in the source program will be assembled but will not be output on the assembly list.

Use

- o Use the NOLIST control instruction to merely control the amount of assembly list output.
- o Use the LIST control instruction to release the assembly list output suppression specified by the NOLIST control instruction.

  By using a combination of NOLIST and LIST control instructions, you may control the amount of assembly list output as well as the contents of the list.

Explanation

- o The NOLIST control instruction functions to suppress assembly list output and is not intended to stop the assembly process.

- o If the LIST control instruction is specified after the NOLIST control instruction, statements described after the LIST control instruction will be output again on the assembly list. The image of the LIST or NOLIST control instruction will also be output to the assembly list.
- o If neither the LIST nor NOLIST control instruction is specified, all statements in the source module will be output to an assembly list.

Application Example

```
        NAME    SAMP1
$       NOLIST          ;(1)
DATA1   EQU     10H
DATA2   EQU     11H
          :
DATAX   EQU     20H
$       LIST            ;(2)
        CSEG
          :
        END
```

Statements in this part will not be output to the assembly list.

(1) Because the NOLIST control instruction is specified here, statements after "$ NOLIST" and up to the LIST control instruction in (2) will not be output on the assembly list. The image of the NOLIST control instruction itself will be output on the list.

(2) Because the LIST control instruction is specified here, statements after this control instruction will be output again on the assembly list. The image of the LIST control instruction will also be output on the list.

(3) GEN/NOGEN (generate/no generate)

Description Format

```
[△]$[△]GEN
[△]$[△]NOGEN
```

Function

- o The GEN control instruction tells the assembler to output macrodefinition lines, macro reference lines, and macro-expanded lines to an assembly list.
- o The NOGEN control instruction tells the assembler to output macrodefinition lines and macro reference lines but suppress the output of macro-expanded lines.

Use

Use the GEN/NOGEN control instruction to control the amount of assembly list output.

Explanation

- o If neither the GEN nor the NOGEN control instruction is specified, the assembler will assume GEN and output macrodefinition lines, macro reference lines, and macro-expanded lines.
- o The specified list control takes place after the image of the GEN or NOGEN control instruction itself has been printed on the assembly list.
- o The assembler continues its processing and increment the STNO count even after the list output control by the NOGEN control instruction.
- o If the GEN control instruction is specified after the NOGEN control instruction, the assembler will resume the output of macro-expanded lines.

Application Example

<Source program>

```
         NAME     SAMP
$        NOGEN
ADMAC    MACRO    PARA1, PARA2
         MOV      A, #PARA1
         ADD      A, #PARA2
         ENDM
         CSEG
         ADMAC    10H, 20H
         END
```

When the above source program is assembled, the output assembly list will look like this.

```
         NAME     SAMP
$        NOGEN
ADMAC    MACRO    PARA1, PARA2
         MOV      A, #PARA1
         ADD      A, #PARA2
         ENDM
         CSEG
         ADMAC    10H, 20H
         MOV      A, #10H
         ADD      A, #20H
         END
```

Macro-expanded part will not be output.

(1) Because NOGEN control instruction is specified, the macro-expanded lines will not be output to the list.

(4) COND/NOCOND (condition/no condition)

Description Format

```
[△]$[△]COND
[△]$[△]NOCOND
```

Function

- o The COND control instruction indicates to the assembler the output of lines which have satisfied the conditional assembly condition and those which have not satisfied the conditional assembly condition to an assembly list.
- o The NOCOND control instruction indicates to the assembler the output of only lines which have satisfied the conditional assembly condition to an assembly list. The output of lines which have not satisfied the conditional assembly condition and lines in which IF/_IF, ELSEIF, _ELSEIF, ELSE, and ENDIF have been described will be suppressed.

Use

Use the COND/NOCOND control instruction to control the amount of assembly list output.

Explanation

- o If neither the COND nor the NOCOND control instruction is specified, the assembler will assume COND and output lines which have satisfied the conditional assembly condition and those which have not satisfied the conditional assembly condition to an assembly list.
- o The specified list control takes place after the image of the COND or NOCOND control instruction itself has been printed on the assembly list.
- o The assembler increments the ALNO and STNO counts even after the list output control by the NOCOND control instruction.

- o If the COND control instruction is specified after the NOCOND control instruction, the assembler will resume the output of lines which have not satisfied the conditional assembly condition and lines in which IF/_IF, ELSEIF, _ELSEIF, ELSE, and ENDIF have been described.

Application Example

<Source program>

```
        NAME      SAMP

$       NOCOND
$       SET (SW1)

$       IF (SW1)

        MOV       A, #1H
$       ELSE
        MOV       A, #0H
        ENDIF

        END
```

This part, though assembled, will not be output to the list.

(5) TITLE (title)

Description Format

```
[△]$[△]TITLE[△]([△]'title-string'[  ])
[△]$[△]TT[△]([△]'title-string'[△])    ; Abbreviated format
```

Function

The TITLE control instruction specifies the character string to be printed in the TITLE column (i.e., title string) at each page header of an assembly list, symbol table list, or cross-reference list.

Use

- o Use the TITLE control instruction to print a title on each page of a list so that the contents of the list can be readily identified.
- o If you are to specify a title for each list in the start-up command line with the assembler option, use this control instruction in the source module file and then you can save your time and labor in starting up the assembler.

Explanation

- o The TITLE control instruction can be described only in the header section of a source module file.
- o If two or more TITLE control instructions are specified at the same time, the assembler will give precedence to the last specified control instruction.
- o Up to 60 characters can be specified as the title string. If the specified title string consists of 61 or more characters, the assembler will accept only the first 60 characters of the string as valid. However, if the character length specification (X) per line of an assembly list file is 119 characters (117 characters with 78K/III) or less, "X - 60 characters" ("X - 58 characters" with 78K/III) will be the acceptable title string length.

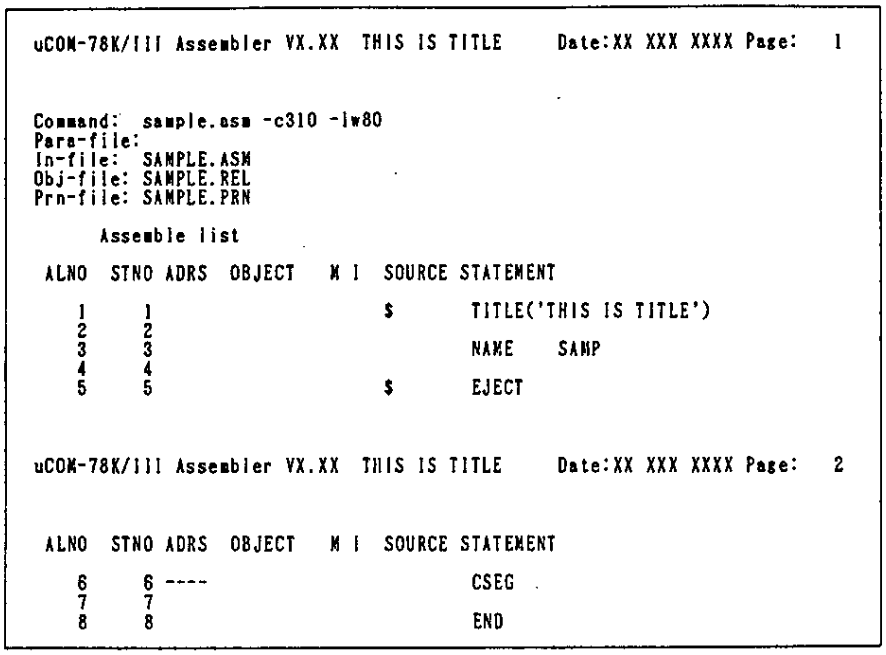
- o If a single quote (') is to be used in the title string as it is originally intended, describe the single quote twice in succession.
- o If no title string is specified (the number of characters in the title string = 0), the assembler will leave the TITLE column blank.
- o If any character not included in 2.2.2, Character set is found in the specified title string, the assembler will output "!" in place of the illegal character in the TITLE column.
- o A title for an assembly list can also be specified with the assembler option -LH in the start-up command line of the assembler.

Application Example

<Source module>

```
$     PROCESSOR (310)
$     TITLE ('THIS IS TITLE')

      NAME      SAMP

$     EJECT
      CSEG

      END
```

When the above source program is assembled, the output assembly list will look like this (with the number of lines per page specified as 72).



```
uCOM-78K/III Assembler VX.XX  THIS IS TITLE     Date:XX XXX XXXX Page:   1


Command:  sample.asm -c310 -lw80
Para-file:
In-file:  SAMPLE.ASM
Obj-file: SAMPLE.REL
Prn-file: SAMPLE.PRN

      Assemble list

 ALNO  STNO ADRS  OBJECT   M I  SOURCE STATEMENT

    1     1                     $       TITLE('THIS IS TITLE')
    2     2
    3     3                             NAME    SAMP
    4     4
    5     5                     $       EJECT


uCOM-78K/III Assembler VX.XX  THIS IS TITLE     Date:XX XXX XXXX Page:   2


 ALNO  STNO ADRS  OBJECT   M I  SOURCE STATEMENT

    6     6 ----                        CSEG
    7     7
    8     8                             END
```

(6) SUBTITLE (subtitle)

Description Format

```
[△]$[△]SUBTITLE[△]([△]'character-string'[△])
[△]$[△]ST[△]([△]'character-string'[△])        ; Abbreviated
                                                format
```

Function

The SUBTITLE control instruction specifies the character string to be printed in the SUBTITLE section at each page header of an assembly list.

Use

Use the SUBTITLE control instruction to print a subtitle on each page of an assembly list so that the contents of the assembly list can be readily identified. The character string of a subtitle may be changed for each page.

Explanation

- o Up to 70 characters can be specified as the character string of a subtitle. If the specified subtitle string consists of 71 or more characters, the assembler will accept only the first 70 characters of the string as valid.
- o The character string specified with the SUBTITLE control instruction will be printed in the SUBTITLE section on the page next to the page in which the SUBTITLE control instruction has been specified. However, if the control instruction is specified at the top (first line) of a page, the subtitle will be printed on that page.
- o If the SUBTITLE control instruction is omitted, the SUBTITLE section will be left blank.
- o If a single quote (') is to be used in the subtitle string as it is originally intended, describe the single quote twice in succession.

- o If no subtitle string is specified (the number of characters in the subtitle string = 0), the assembler will leave the SUBTITLE section blank.
- o If any character not included in 2.2.2, Character set is found in the specified subtitle string, the assembler will output "!" in place of the illegal character in the SUBTITLE section. If an CR (0DH) code is described in the subtitle string, the assembler will output an error message and output nothing on the list. If a "00H" code is described, subsequent characters before the closing single quote (') will not be output.

Application Example

<Source module>

```
      NAME      SAMP

      CSEG
        :
        :
$     SUBTITLE('THIS IS SUBTITLE 1')    ;(1)
$     EJECT                             ;(2)

      CSEG
        :
        :
$     SUBTITLE('THIS IS SUBTITLE 2')    ;(3)
$     EJECT                             ;(4)
$     SUBTITLE('THIS IS SUBTITLE 3')    ;(5)
        :
        :
      END
```

(1) This control instruction specifies character string "THIS IS SUBTITLE 1".

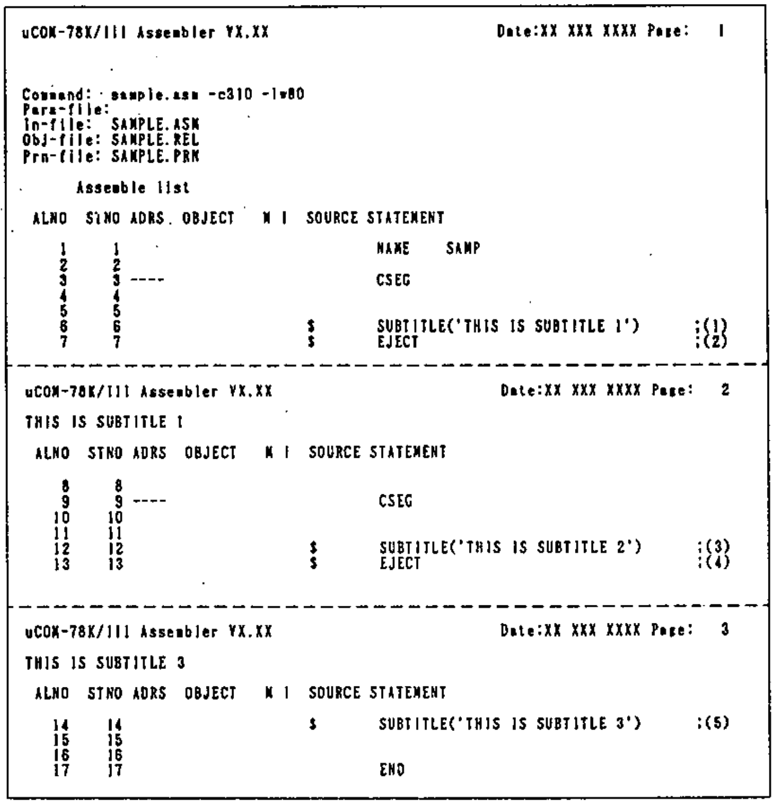
(2) This control instruction indicates page ejection.

(3) This control instruction specifies character string "THIS IS SUBTITLE 2".

(4) This control instruction indicates page ejection.

(5) This control instruction specifies character string "THIS IS SUBTITLE 3".

When the above source program is assembled, the output assembly list will look like this (with the number of lines per page specified as 80).



```
uCOM-78K/III Assembler VX.XX                        Date:XX XXX XXXX Page:   1


Command:  sample.asm -c310 -lw80
Para-file:
In-file:  SAMPLE.ASM
Obj-file: SAMPLE.REL
Prn-file: SAMPLE.PRN

       Assemble list

 ALNO  STNO ADRS  OBJECT   M I  SOURCE STATEMENT

    1     1                          NAME    SAMP
    2     2
    3     3 ----                     CSEG
    4     4
    5     5
    6     6                 $        SUBTITLE('THIS IS SUBTITLE 1')     ;(1)
    7     7                 $        EJECT                              ;(2)
---------------------------------------------------------------------------
uCOM-78K/III Assembler VX.XX                        Date:XX XXX XXXX Page:   2

THIS IS SUBTITLE 1

 ALNO  STNO ADRS  OBJECT   M I  SOURCE STATEMENT

    8     8
    9     9 ----                     CSEG
   10    10
   11    11
   12    12                 $        SUBTITLE('THIS IS SUBTITLE 2')     ;(3)
   13    13                 $        EJECT                              ;(4)

---------------------------------------------------------------------------
uCOM-78K/III Assembler VX.XX                        Date:XX XXX XXXX Page:   3

THIS IS SUBTITLE 3

 ALNO  STNO ADRS  OBJECT   M I  SOURCE STATEMENT

   14    14                 $        SUBTITLE('THIS IS SUBTITLE 3')     ;(5)
   15    15
   16    16
   17    17                          END
```

←Page ejection by instruction in (2)

←Subtitle printing by instruction in (1)

←Page ejection by instruction in (4)

←Subtitle printing by instruction in (5) because subtitle (5) is at the 1st line.

4.7 Conditional Assembly Control Instructions

Conditional assembly control instructions select a series of statements in a source module as those subject to assembly or not subject to assembly by setting switches for conditional assembly. Conditional assembly control instructions are available in two groups: one group to set the condition for limiting source statements subject to assembly (IF/_IF, ELSEIF/_ELSEIF, ELSE, and ENDIF) and the other, to give a true or false value to a specified switch name (SET and RESET).

By making the best of these control instructions, assembly of a source module by excluding unwanted statements can be executed with little or no change to the source module.

(1) IF/_IF, ELSEIF/_ELSEIF, ELSE, ENDIF

Description Format

```
[△]$[△]IF[△]([△]switch-name[[△]:[△]switch-name]...[△])
or [△]$[△]_IF△conditional-expression
[△]$[△]ELSEIF[△]([△]switch-name[[△]:[△]switch-name]...[△])
or [△]$[△]_ELSEIF△conditional-expression
[△]$[△]ELSE
[△]$[△]ENDIF
```

Function

- o These control instructions set the conditions to limit source statements subject to conditional assembly and those not subject to conditional assembly.
  Source statements described between the IF or _IF control instruction and the ENDIF control instruction are subject to conditional assembly.
- o If the evaluated value of the switch name or conditional expression specified by the IF or _IF control instruction (i.e., IF or _IF condition) is true, source statements described after this IF or _IF control instruction until the appearance of the next conditional assembly control instruction (ELSEIF/_ELSEIF, ELSE, or ENDIF) in the source program will be assembled. For subsequent assembly processing, the assembler will proceed to the statement next to the ENDIF control instruction. If the IF or _IF condition is false, source statements described after this IF or _IF control instruction until the appearance of the next conditional assembly control instruction (ELSEIF/_ELSEIF, ELSE, or ENDIF) in the source program will not be assembled.

- The **ELSEIF** or **_ELSEIF** control instruction is checked for true/false only when the conditions of all the conditional assembly control instructions described before this ELSEIF or _ELSEIF control instruction are not satisfied (i.e., all the evaluated values of the switch names or conditional expressions are false).
  If the evaluated value of the switch name or conditional expression specified by the ELSEIF or _ELSEIF control instruction (i.e., ELSEIF or _ELSEIF condition) is true, source statements described after this ELSEIF or _ELSEIF control instruction until the appearance of the next conditional assembly control instruction (ELSEIF/_ELSEIF, ELSE, or ENDIF) in the source program will be assembled. For subsequent assembly processing, the assembler will proceed to the statement next to the ENDIF control instruction.
  If the ELSEIF or _ELSEIF condition is false, source statements described after this ELSEIF or _ELSEIF control instruction until the appearance of the next conditional assembly control instruction (ELSEIF/_ELSEIF, ELSE, or ENDIF) in the source program will not be assembled.
- If the conditions of all the IF/_IF and ELSEIF/_ELSEIF control instructions described before the **ELSE** control instruction are not satisfied (i.e., all the evaluated values of the switch names or conditional expressions are false), source statements described after this ELSE control instruction until the appearance of the ENDIF control instruction in the source program will be assembled.
- The **ENDIF** control instruction indicates to the assembler the termination of source statements subject to conditional assembly.

Use

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.

- If a statement for debugging necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

Explanation

- The IF and ELSEIF control instructions are used for true/false condition judgment with switch name(s), whereas the _IF and _ELSEIF control instructions are used for true/false condition judgment with a conditional expression.
- With the IF and ELSEIF control instructions, at least one switch name must be described.
  The rules of describing switch names are the same as the conventions of symbol description, for which see Subsection 2.2.3 (1), "Symbol field" in Chapter 2. However, the maximum number of characters that can be recognized as a switch name is always 8.
- If two or more switch names are to be specified with the IF or ELSEIF control instruction, delimit each switch name with a colon (:). Up to five switch names can be used per module.
- When two or more switch names have been specified with the IF or ELSEIF control instruction, the IF or ELSEIF condition is judged as satisfied if one of the switch name values is true.
- The value of each switch name to be specified with the IF or ELSEIF control instruction must be defined with the SET or RESET control instruction. (See (2), "SET, RESET" in this section.) Therefore, the value of the switch name specified with the IF or ELSEIF control instruction must have been set in the source module with the SET or RESET control instruction.
- If the specified switch name or conditional expression contains an illegal description, the assembler will output an error message and determine that the evaluated value is false.

- When describing the IF or _IF control instruction, the IF or _IF control instruction must always be paired with the ENDIF control instruction.
- If an IF-ENDIF block is described in a macro body and control is transferred back from the macro at that level by the EXITM processing, the assembler will force the IF level to return to that level at the entry of the macro body. In this case, no error will result.
- Description of an IF-ENDIF block in another IF-ENDIF block is referred to as nesting IF control instructions. Nesting of IF control instructions is allowed up to 8 levels.
- In conditional assembly, object codes will not be generated for statements not assembled, but these statements will be output without change on the assembly list. If you do not wish to output these statements, use the NOCOND control instruction.

Application Example

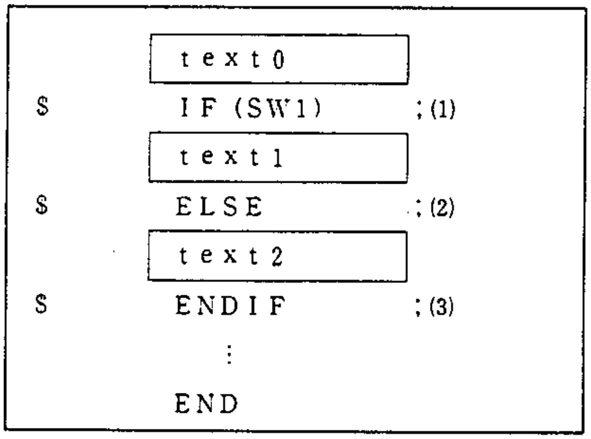
Example 1

```
            ┌──────────────┐
            │ text0        │
            └──────────────┘
$             IF (SW1)        ;(1)
            ┌──────────────┐
            │ text1        │
            └──────────────┘
$             ENDIF           ;(2)
                :
              END
```

(1) If the value of switch name "SW1" is true (00FFH), statements in "text1" will be assembled.
If the value of switch name "SW1" is false (0000H), statements in "text1" will not be assembled.
The value of switch name "SW1" has been set to true (00FFH) or false (0000H) with the SET or RESET control instruction described in "text0".

(2) This instruction indicates the end of the source statement range for conditional assembly.

Example 2



```
              text0
$             IF (SW1)        ;(1)
              text1
$             ELSE            ;(2)
              text2
$             ENDIF           ;(3)
                :
              END
```

(1) The value of switch name "SW1" has been set to true (00FFH) or false (0000H) with the SET or RESET control instruction described in "text0".
If the value of switch name "SW1" is true (00FFH), statements in "text1" will be assembled and statements in "text2" will not be assembled.

(2) If the value of switch name "SW1" in (1) is false (0000H), statements in "text1" will not be assembled and statements in "text2" will be assembled.

(3) This instruction indicates the end of the source statement range for conditional assembly.

Example 3

```
              text0
$             IF (SW1)              ;(1)
              text1
$             ELSEIF (SW2)          ;(2)
              text2
$             ELSEIF (SW3)          ;(3)
              text3
$             ELSE                  ;(4)
              text4
$             ENDIF                 ;(5)
                 :
              END
```

(1) The values of switch names "SW1", "SW2", and "SW3" have been set to true (00FFH) or false "0000H" with the SET or RESET control instruction described in "text0".
If the value of switch name "SW1" is true, statements in "text1" will be assembled and statements in "text2", "text3", and "text4" will not be assembled.
If the value of switch name "SW1" is false, statements in "text1" will not be assembled and conditional assembly of statements in "text2" and thereafter will be executed.

(2) If the value of switch name "SW1" in (1) is false and the value of switch name "SW2" is true, statements in "text2" will be assembled and statements in "text1", "text3", and "text4" will not be assembled.

(3) If the values of both switch names "SW1" in (1) and "SW2" in (2) are false and the value of switch name "SW3" is true, statements in "text3" will be assembled and statements in "text1", "text2" and "text4" will not be assembled.

(4) If the values of switch names "SW1" in (1), "SW2" in (2), and "SW3" in (3) are all false, statements in "text4" will be assembled and statements in "text1", "text2" and "text3" will not be assembled.

(5) This instruction indicates the end of the source statement range for conditional assembly.

Example 4

```
             text0
$         IF (SWA:SWB)         ;(1)
             text1
$         ENDIF                ;(2)
            :
          END
```

(1) The values of switch names "SWA" and "SWB" have been set to true (00FFH) or false "0000H" with the SET or RESET control instruction described in "text0".
If the value of switch name "SWA" or "SWB" is true, statements in "text1" will be assembled.
If the values of both switch names "SWA" and "SWB" are false, statements in "text1" will not be assembled.

(2) This instruction indicates the end of the source statement range for conditional assembly.

(2) SET, RESET (set, reset)

Description Format

```
[△]$[△]SET[△]([△]switch-name[[△]:[△]switch-name]...[△])
[△]$[△]RESET[△]([△]switch-name[[△]:[△]switch-name]...[△])
```

Function

- o The SET and RESET control instructions give a value to each switch name to be specified with the IF or ELSEIF control instruction.
- o The SET control instruction gives a true value (00FFH) to each switch name specified in its operand.
- o The RESET control instruction gives a false value (0000H) to each switch name specified in its operand.

Use

- o Describe the SET control instruction to give a true value (00FFH) to each switch name to be specified with the IF or ELSEIF control instruction.
- o Describe the RESET control instruction to give a false value (0000H) to each switch name to be specified with the IF or ELSEIF control instruction.

Explanation

- o With the SET and RESET control instructions, at least one switch name must be described.
  The rules of describing switch names are the same as the conventions of symbol description, for which see Subsection 2.2.3 (1), "Symbol field" in Chapter 2. However, the maximum number of characters that can be recognized as a switch name is always 8.
- o If two or more switch names are to be specified with the SET or RESET control instruction, delimit each switch name with a colon (:). Up to five switch names can be used per module.

- o The specified switch name(s) may be the same as user-defined symbol(s) other than reserved words and other switch names.
- o The switch name once set to "true" with the SET control instruction can be changed to "false" with the RESET control instruction, and vice versa.
- o A switch name to be specified with the IF or ELSEIF control instruction must be defined at least once with the SET or RESET control instruction in the source module before describing the IF or ELSEIF control instruction.
- o Switch names will not be output to a cross-reference list.

Application Example

Example 1

```
          :
$         SET (SW1)               ;(1)
          :
$         IF (SW1)                ;(2)
            text1
$         ENDIF                   ;(3)
          :
$         RESET (SW1 : SW2)       ;(4)
          :
$         IF (SW1)                ;(5)
            text2
$         ELSEIF (SW2)            ;(6)
            text3
$         ELSE                    ;(7)
            text4
$         ENDIF                   ;(8)
            :
          END
```

(1) This instruction gives a true value (00FFH) to switch name "SW1".

(2) Because the true value has been given to switch name "SW1" in (1) above, statements in "text1" will be assembled.

(3) This instruction indicates the end of the source statement range for conditional assembly, which starts from (2).

(4) This instruction gives a false value (0000H) to switch names "SW1" and "SW2", respectively.

(5) Because the false value has been given to switch name "SW1" in (4) above, statements in "text2" will not be assembled.

(6) Because the false value has also been given to switch name "SW2" in (4) above, statements in "text3" will not be assembled.

(7) Because both switch names "SW1" and "SW2" are false in (5) and (6) above, statements in "text4" will be assembled.

(8) This instruction indicates the end of the source statement range for conditional assembly, which starts from (5).

CHAPTER 5. MACROS

5.1 Overview of Macro

When you must describe a series of instruction groups over and over again in a source program, a macro function is very useful for program description.

The macro function refers to the expansion of a series of instruction groups defined as a macro body with MACRO and ENDM directives into the location where the macro name is referenced.

A macro is used to increase coding efficiency of a source program and is different from a subroutine.

A macro and a subroutine each have the following features and should be used selectively according to the specific purpose.

(1) Subroutine

- o Describe a process (or the same sequence of instructions) which must be repeated over and over again in a program as a subroutine. The subroutine will be converted into machine language just once by the assembler.
- o To call the subroutine, you only need to describe a subroutine call instruction. (Generally, instructions to set arguments are also described before and after the subroutine.)

  Therefore, by making the best of subroutines, the program memory can be used with high efficiency.
- o By coding a series of processes in a program as subroutines, the program can be structurized. (By this structurization, the programmer can easily understand the overall structure of the program, thus making the program design easy.)

(2) Macro

- o The basic function of a macro is the replacement of a group of instructions with a name.

  A series of instruction groups defined as a macro body with MACRO and ENDM directives will be expanded into the location where the macro name is referenced.
- o When the assembler detects a macro reference, the assembler expands the macro body and converts the group of instructions into machine language while replacing the formal parameter(s) of the macro body with the actual parameters at the time of the macro reference.

- Parameters can be described for a macro.
  For example, if there are instruction groups which are the same in processing procedure but are different in the data to be described in the operand, define a macro by assigning formal parameter(s) to the data. By describing the macro name and the actual parameter(s) at macro reference time, the assembler can cope with various instruction groups which differ only in part of the statement description.

The programming technique with subroutines is mainly used for memory size reduction and program structurization, whereas macros are used to increase coding efficiency of the program.

5.2 Utilization of Macros

5.2.1 Macrodefinition

A macro is defined with the MACRO and ENDM directives.

Description Format

Symbol field	Mnemonic field	Operand field	Comment field
macro name	MACRO	[formal parameter[,...]]	[;comment]
	⌇		
	ENDM		

Function

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

Application Example

```
ADMAC   MACRO   PARA1, PARA2
        MOV     A, #PARA1
        ADD     A, #PARA2
        ENDM
```

The above example shows a simple macrodefinition which specifies the addition of two values PARA1 and PARA2 and the storage of the result in register A. The macro is given a name "ADMAC" and "PARA1" and "PARA2" are formal parameters. For details, see (1) MACRO in Section 3.9, Macro Directives, Chapter 3.

5.2.2 Macro reference

To call a macro, the already defined macro name must be described in the Mnemonic field of the source program.

Description Format

Symbol	Mnemonic	Operand	Comment
field	field	field	field
[label:]	macro name	[actual parameter[,...]]	[;comment]

Function

This directive calls the macro body assigned to the macro name specified in the Mnemonic field.

Use

Use this directive description to call a macro body.

Explanation

- The macro name to be specified in the Mnemonic field must have been defined before the macro reference.
- Up to 16 actual parameters may be specified per line by delimiting each actual parameter with a comma (,).
- No Blank character can be described in the character string constituting an actual parameter.
- When describing a comma (,), semicolon (;), Blank, or TAB in an actual parameter, enclose the character string which includes any of these special characters with a pair of single quotes.
- Formal parameters are replaced with their corresponding actual parameters in sequence from left to right.
- A warning error will result if the number of formal parameters is not equal to the number of actual parameters.

Application Example

```
          NAME      SAMPLE
ADMAC     MACRO     PARA1, PARA2
          MOV       A, #PARA1
          ADD       A, #PARA2
          ENDM

          CSEG
           :
          ADMAC     10H, 20H
           :
          END
```

This directive calls the already defined macro name "ADMAC". 10H and 20H are actual parameters.

5.2.3 Macroexpansion

The assembler processes a macro as follows:

- o Expands the macro body corresponding to the referenced macro name to the location where the macro name is referenced.
- o Assembles statements in the expanded macro body just the same as other statements.

Application Example

When the macro referenced in Subsection 5.2.2, "Macro reference" is assembled, the macro body will be expanded as shown below.

```
        NAME    SAMPLE
ADMAC   MACRO   PARA1, PARA2
        MOV     A, #PARA1                    Macrodefinition
        ADD     A, #PARA2
        ENDM

        CSEG
         :
        ADMAC   10H, 20H          ;(1)
        MOV     A, #PARA1  10H
                                             Macroexpansion
        ADD     A, #PARA2  20H
         :
        END
```

By the macro reference in (1), the macro body will be expanded. The formal parameters within the macro body will be replaced with the actual parameters.

5.3 Symbols within Macro

Symbols that can be defined in a macro are divided into two types: global symbols and local symbols.

(1) Global symbols

- o A global symbol is a symbol that can be referenced from any statement within a source program.
  Therefore, if a series of statements are expanded by referencing a macro in which the global symbol has been defined, the symbol will cause a double definition error.
- o Symbols not defined with the LOCAL directive are global.

(2) Local symbols

- o A local symbol is a symbol defined with the LOCAL directive. (See (2) LOCAL in Section 3.9, "Macro directives".)
- o A local symbol can be referenced within the macro declared as LOCAL with the LOCAL directive.
- o No local symbol can be referenced from outside the macro.

Application Example

<Source program>

```
        NAME     SAMPLE
MAC1    MACRO
        LOCAL    LLAB              ;(1)
LLAB:
          :                              Macrodefinition
GLAB:
        BR       $LLAB             ;(2)
        BR       $GLAB             ;(3)
        ENDM
          :
REF1:   MAC1                       ;(4) ←— Macro reference
          :
        BR       !LLAB             ;(5) ←— This description
        BR       !GLAB             ;(6)     is erroneous.
          :
REF2:   MAC1                       ;(7) ←— Macro reference
          :
        END
```

(1) This directive defines label "LLAB" as a local symbol.

(2) This instruction references local symbol "LLAB" in macro MAC1.

(3) This instruction references global symbol "GLAB" in macro MAC1.

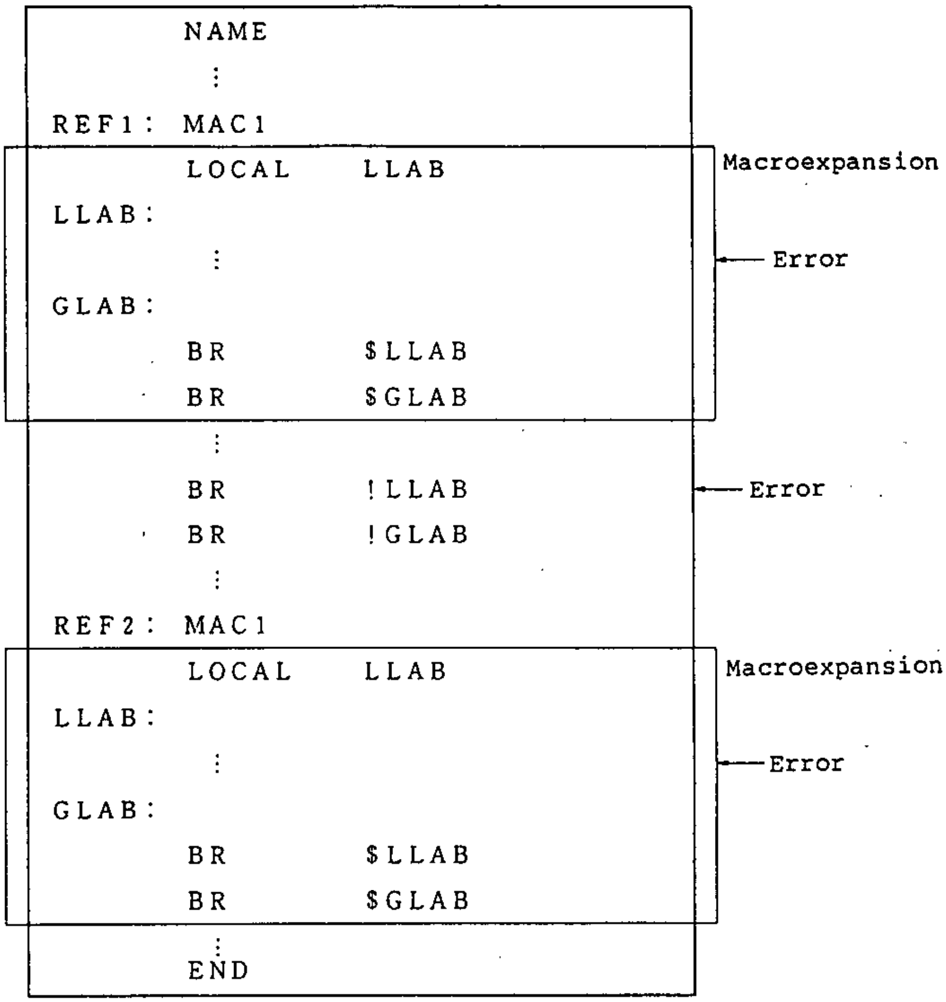
(4) This directive references macro MAC1.

(5) This instruction references local symbol "LLAB" from outside the definition of macro MAC1. This description cause an error when the source program is assembled.

(6) This instruction references global symbol "GLAB" from outside the definition of macro MAC1.

(7) This directive references macro MAC1. The same macro is referenced twice.

When the source program in the above program is assembled, the macro body will be expanded as shown below.



```
        NAME
          :
REF1:   MAC1
        LOCAL     LLAB
LLAB:
          :
GLAB:
        BR        $LLAB
        BR        $GLAB
          :
        BR        !LLAB
        BR        !GLAB
          :
REF2:   MAC1
        LOCAL     LLAB
LLAB:
          :
GLAB:
        BR        $LLAB
        BR        $GLAB
          :
        END
```

Global symbol "GLAB" has been defined in macro MAC1. Because macro MAC1 is referenced twice, global symbol "GLAB" causes a double definition error as a result of expanding a series of statements in the macro body.

5.4 Macro Operators

Two types of macro operators are available: "& (Concatenate)" and "' (single quote)".

(1) & (Concatenate)

- The concatenating sign "&" concatenates one character string to another within a macro body. At macroexpansion time, the character string on the left of the concatenating sign is concatenated to the character string on the right of the sign. The "&" sign itself disappears after concatenating the strings.
- At macrodefinition time, a string before or after "&" in a symbol can be recognized as a formal parameter or LOCAL symbol. At macroexpansion time, the formal parameter or LOCAL symbol before or after "&" is evaluated as a symbol and can be concatenated in the symbol.
- The "&" sign enclosed in a pair of single quotes is handled as mere data.
- Two "&" signs described in succession are handled as a single "&" sign.

Example:

Macrodefinition

```
M1          MACRO     X
LAB&X:      PUSH      R0
            D&B       10H
            DB        'X'
            DB        X
            DB        '&X'
            ENDM
```

←Formal parameter "X" is recognized.

Macroreference

```
            M1        1
LAB1:       PUSH      RO
            DB        10H
            DB        'X'
            DB        1
            DB        '&X'
```

←D and B are concatenated and become "DB".

←& enclosed in a pair of single quotes is handled as mere data.

(2) ' (Single quote)

- o If a character string enclosed in a pair of single quotes is described at the beginning of an actual parameter in a macrodefinition or an IRP directive or after a delimiting character, the character string will be interpreted as an actual parameter. The character string will be passed to the actual parameter without the enclosing single quotes.
- o If a character string enclosed in a pair of single quotes exists in a macro body, the character string will be handled as mere data.
- o To use a single quote as it is originally intended (as a single quotation mark), describe the single quote twice in succession.

Example:

```
MAC1      MACRO       X
          IRP         Z,<X>
          MOV         A, #Z
          ENDM
          ENDM
           :
          MAC1        '10, 20, 30'
```

When the source program in the above program is assembled, MAC1 will be expanded as shown below.

```
          IRP         Z.<10, 20, 30>
          MOV         A, #Z
          ENDM
          MOV         A, #10          ┐
          MOV         A, #20          │ Expansion of IRP
          MOV         A, #30          ┘
```

CHAPTER 6. PRODUCT UTILIZATION

There are several ways to effective use this package for assembly of source modules. Only a few of these techniques are introduced in this section.

(1) How to save your trouble in starting up the assembler

It is better to describe in a source module file, control instructions which have the same functions as assembler options and which you must always use when starting up the assembler such as the processor type specification (-C) and debug information output specification (-G). Especially, the processor type specification which cannot be omitted should be specified in the module header using the PROCESSOR control instruction. Then, you do not need to specify the assembler option (-C) in the start-up command line each time you start up the assembler program. An error will result if you forget to specify this assembler option in the start-up command line and you must start up the assembler again from the beginning with the correct assembler options.

The cross-reference list output control instruction (XREF) should also be specified in the module header.

Example

```
$     PROCESSOR(312)
$     DEBUG
$     XREF

      NAME       TEST

      CSEG
       :
```

(2) How to develop programs with high memory utilization efficiency

The short direct addressing area is an area which can be accessed with instructions of short byte length as compared with other data memory areas.

Therefore, by using this area efficiently, a program with high memory utilization efficiency can be developed.

So, if you declare the short direct addressing area with one module and if all the variables which you intended to locate in the short direct addressing area cannot be located, you can make changes easily so that only variables to be accessed frequently are located in the short direct addressing area.

Module 1

```
                PUBLIC          TMP1, TMP2
WORK            DSEG            AT  0FE20H
TMP1:           DS              2   ; word
TMP2:           DS              1   ; byte
                  :
```

Module 2

```
                EXTRN           TMP1, TMP2
SUB             CSEG
                MOVW            TMP1, #1234H
                MOV             TMP2, #56H
                  :
```

APPENDIX A. LIST OF RESERVED WORDS

Reserved words are available in six types: machine language instructions, directives, control instructions, operators, register names, and sfr symbols. The reserved words are character strings reserved beforehand by the assembler and cannot be used for other than the intended purposes.
Types of reserved words that can be described in each field of a source program is shown below.

Symbol field	All reserved words cannot be described in this field.
Mnemonic field	Only machine language instructions and directives can be described in this field.
Operand field	Only operators, sfr symbols, and register names can be described in this field.
Comment field	All reserved words can be described in this field.

Reserved words for each microcomputer in the 78K series are listed in Sections A.1 through A.4, respectively, in this appendix.

A.1 List of Reserved Words for 78K/0

Instructions	ADD	ADDC	ADDW	ADJBA	ADJBS	ALU	AND
	AND1	BC	BF	BNC	BNZ	BR	BRK
	BT	BTCLR	BZ	CALL	CALLF	CALLT	CLR1
	CLR1	CMP	CMPW	DBNZ	DEC	DECW	DI
	DIVUW	EI	HALT	INC	INCW	MOV	MOV1
	MOVW	MOVW	MULU	NOP	NOT1	OR	OR1
	POP	PUSH	RET	RETB	RETI	ROL	ROL4
	ROLC	ROR	ROR4	RORC	SEL	SET1	SET1
	STOP	SUB	SUBC	SUBW	XCH	XCHW	XOR
	XOR1						
Operators	AND	EQ	GE	GT	HIGH	LE	LOW
	LT	MOD	NE	NOT	OR	SHL	SHR
	XOR						
Directives	BR	BSEG	CSEG	DB	DBIT	DS	DSEG
	DW	END	ENDM	ENDS	EQU	EXITM	EXTBIT
	EXTRN	IRP	LOCAL	MACRO	NAME	ORG	PUBLIC
	REPT	SET					
Control instructions	COND	DEBUG	DG	EJ	EJECT	ELSE	ELSEIF
	_ELSEIF	ENDIF	GEN	IC	IF	_IF	INCLUDE
	LI	LIST	NOCOND	NODEBUG	NODG	NOGEN	NOLI
	NOLIST	NOXR	NOXREF	PC	PROCESSOR	RESET	SET
	SUBTITLE	ST	TITLE	TT	XR	XREF	
sfr symbols	ADCR	ADIS	ADM	ADTC	ADTP	CKM	CR00
	CR01	CR10	CR20	CSIM0	CSIM1	IF0	IF0H
	IF0L	IMS	INTM0	KRM	MK0	MK0H	MK0L
	MM	OSTS	P0	P1	P2	P3	P4
	P5	P6	PCC	PM0	PM1	PM2	PM3
	PM5	PM6	PR0	PR0H	PR0L	PSW	PU0
	SBIC	SCS	SI00	SI01	SINT	SP	SVA
	TCL0	TCL1	TCL2	TCL3	TM0	TM1	TM2
	TMC0	TMC1	TMC2	TOC0	TOC1	WDTM	
Registers	A	AX	B	BC	C	CY	D
	DE	E	H	HL	L	PSW	R0
	R1	R2	R3	R4	R5	R6	R7
	RB0	RB1	RB2	RB3	RP0	RP1	RP2
	RP3	SP	X				
Seg. attr.	AT	CALLT	FIXED	IHRAM	SADDR	SADDRP	UNIT

* Seg. Attr.: Segment attributes

A.2 List of Reserved Words for 78K/I

Instructions	μPD78112						
	ADD	ADDC	ADDW	ADJBA	ADJBS	AND	AND1
	BC	BE	BF	BL	BNC	BNE	BNL
	BNZ	BR	BT	BTCLR	BZ	CALL	CALLF
	CALLT	CLR1	CLR1	CMP	CMPW	DBNZ	DEC
	DECW	DI	DIVUW	EI	INC	INCW	MOV
	MOV	MOV	MOV1	MOVW	MULUW	NOP	NOT1
	NOT1	OR	OR1	POP	PUSH	RET	RETI
	ROL	ROL4	ROLC	ROR	ROR4	RORC	SEL
	SET1	SET1	SHL	SHR	SHRL	SHRW	SUB
	SUBC	SUBW	XCH	XOR	XOR1		
	μPD78134						
	ADD	ADDC	ADDW	ADJBA	ADJBS	AND	AND1
	BC	BE	BF	BL	BNC	BNE	BNL
	BNZ	BR	BT	BTCLR	BZ	CALL	CALLF
	CALLT	CLR1	CLR1	CMP	CMPW	DBNZ	DEC
	DECW	DI	DIVUW	EI	INC	INCW	MOV
	MOV	MOV1	MOVW	MOVW	MULUW	NOP	NOT1
	NOT1	OR	OR1	POP	PUSH	RET	RETI
	ROL	ROL4	ROLC	ROR	ROR4	RORC	SEL
	SET1	SET1	SHL	SHLW	SHR	SHRW	SUB
	SUBC	SUBW	XCH	XOR	XOR1		
	μPD78136/μPD78138						
	ADD	ADDC	ADDW	ADJBA	ADJBS	AND	AND1
	BC	BE	BF	BL	BNC	BNE	BNL
	BNZ	BR	BT	BTCLR	BZ	CALL	CALLF
	CALLT	CLR1	CLR1	CMP	CMPW	DBNZ	DEC
	DECW	DI	DIVUW	EI	INC	INCW	MOV
	MOV	MOV1	MOVW	MOVW	MULSW	MULUW	NOP
	NOT1	NOT1	OR	OR1	POP	PUSH	RET
	RETI	ROL	ROL4	ROLC	ROR	ROR4	RORC
	SEL	SET1	SET1	SHL	SHLW	SHR	SHRW
	SUB	SUBC	SUBW	XCH	XOR	XOR1	
Operators	AND	EQ	GE	GT	HIGH	LE	LOW
	LT	MOD	NE	NOT	OR	SHL	SHR
	XOR						

Directives	BR	BSEG	CSEG	DB	DBIT	DS	DSEG
	DW	END	ENDM	ENDS	EQU	EXITM	EXTBIT
	EXTRN	IRP	LOCAL	MACRO	NAME	ORG	PUBLIC
	REPT	SET					
Control instructions	COND	DEBUG	DG	EJ	EJECT	ELSE	ELSEIF
	_ELSEIF	ENDIF	GEN	IC	IF	_IF	INCLUDE
	LI	LIST	NOCOND	NODEBUG	NODG	NOGEN	NOLI
	NOLIST	NOXR	NOXREF	PC	PROCESSOR	RESET	SET
	SUBTITLE	ST	TITLE	TT	XR	XREF	
sfr symbols	μPD78112						
	ADM	CPT0	CPT1	CPT2	CPT3	CPTM	CR00
	CR01	CR02	CR10	CR11	CR12	CR20	CSIM
	EDVC	FRC	ICR	IF0	IF0H	IF0L	INTM0
	INTM1	ISM0	ISM0H	ISM0L	MK0	MK0H	MK0L
	MM	P1	P2	P3	P4	P5	P6
	PM1	PM3	PM5	PM6	PMC3	PSW	PWM0
	PWM1	PWMC	SA	SIO	SP	STBC	TM0
	TM1	TM2	TMC0	TMC1			
	μPD78134/μPD78136/μPD78138						
	ADCR	ADM	CLOM	CPT0	CPT1	CPT2H	CPT2L
	CPT3	CPT30	CRTM	CR00	CR01	CR02	CR10
	CR11	CR12	CR20	CR30	CSIM	EC	ECC0
	ECC1	EDVC	FRC	ICR	IF0	IF0H	IF0L
	IMS	INTM0	INTM1	ISM0	ISM0H	ISM0L	IST
	MK0	MK0H	MK0L	MM	P0	P0H	P0L
	P1	P2	P3	P4	P5	P6	P7
	PM0	PM1	PM3	PM5	PM6	PM7	PMC3
	PR0	PR0H	PR0L	PRM3	PSW	PU0	PWM0
	PWM1	PWMC	RTPC	SBIC	SI0	SP	STBC
	TM0	TM1	TM2	TM3	TMC0	TMC1	TOC0
	TOC1	TOM0	TOM1				
Registers	A	AX	B	BC	C	CY	D
	DE	E	H	HL	L	R0	R1
	R2	R3	R4	R5	R6	R7	RP0
	RP1	RP2	RP3	RB0	RB1	RB2	RB3
	X						
Seg. attr.	AT	CALLT0	FIXED	SADDR	UNIT		

*Seg. attr.: Segment attributes

A.3 List of Reserved Words for 78K/III

Instructions	μPD78310/μPD78312/μPD78310A/μPD78312A						
	ADD	ADDC	ADDW	ADJ4	AND	AND1	BC
	BE	BF	BFSET	BGE	BGT	BH	BL
	BLE	BLT	BN	BNC	BNE	BNH	BNL
	BNV	BNZ	BP	BPE	BP0	BR	BRK
	BRKCS	BT	BTCLR	BV	BZ	CALL	CALLF
	CALLT	CLR1	CMP	CMPBKC	CMPBKE	CMPBKN	CMPBKNE
	CMPMC	CMPME	CMPMNC	CMPMNE	CMPW	DBNZ	DEC
	DECW	DI	DIVUW	DIVUX	EI	INC	INCW
	MOV	MOVBK	MOVM	MOVW	MOV1	MULU	MULUW
	NOP	NOT1	OR	OR1	POP	POPU	PUSH
	PUSHU	RET	RETCS	RETI	ROL	ROLC	ROR
	RORC	ROL4	ROR4	SEL	SET1	SHL	SHR
	SHLW	SHRW	SUB	SUBC	SUBW	SWRS	XCH
	XCHBK	XCHW	XOR	XOR1			
	μPD78320/μPD78322/μPD78327/μPD78328/μPD78330/μPD78334						
	ADD	ADDC	ADDW	ADJ4	ADJBA	ADJBS	AND
	AND1	BC	BE	BF	BFSET	BGE	BGT
	BH	BL	BLE	BLT	BN	BNC	BNE
	BNH	BNL	BNV	BNZ	BP	BPE	BP0
	BR	BRK	BRKCS	BT	BTCLR	BV	BZ
	CALL	CALLF	CALLT	CHKL	CHKLA	CLR1	CMP
	CMPBKC	CMPBKE	CMPBKNC	CMPBKNE	CMPMC	CMPME	CMPMNC
	CMPMNE	CMPW	CVTBW	DBNZ	DEC	DECW	DI
	DIVUW	DIVUX	EI	INC	INCW	MOV	MOVBK
	MOVM	MOVW	MOV1	MULU	MULUW	MULW	NOP
	NOT1	OR	OR1	POP	POPU	PUSH	PUSHU
	RET	RETB	RETCS	RETCSB	RETI	ROL	ROLC
	ROR	RORC	ROL4	ROR4	SEL	SET1	SHL
	SHR	SHLW	SHRW	SUB	SUBC	SUBW	SWRS
	XCH	XCHBK	XCHM	XCHW	XOR	XOR1	
Operators	AND	EQ	GE	GT	HIGH	LE	LOW
	LT	MOD	NE	NOT	OR	SHL	SHR
	XOR						
Directive	BR	BSEG	CSEG	DB	DBIT	DS	DSEG
	DW	END	ENDM	ENDS	EQU	EXITM	EXTBIT
	EXTRN	IRP	LOCAL	MACRO	NAME	ORG	PUBLIC
	REPT	RSS	SET				

Control instructions	COND	DEBUG	DG	EJ	EJECT	ELSE	ELSEIF
	_ELSEIF	ENDIF	GEN	IC	IF	_IF	INCLUDE
	LI	LIST	NOCOND	NODEBUG	NODG	NOGEN	NOLI
	NOLIST	NOXR	NOXREF	PC	PROCESSOR	RESET	SET
	SUBTITLE	ST	TITLE	TT	XR	XREF	
sfr symbols	µPD78310/µPD78312/µPD78310A/µPD78312A						
	ADCR	ADIC	ADM	ADMS	BRG	CCW	CPT
	CPT0H	CPT0L	CPT1	CPT1H	CPT1L	CPTM	CR00
	CR00H	CR00L	CR01	CR01H	CR01L	CR10	CR10H
	CR10L	CR11	CR11H	CR11L	CRC	CRIC00	CRIC01
	CRIC10	CRIC11	CRMS00	CRMS10	CUIM	EXIC0	EXIC1
	EXIC2	EXMS0	EXMS1	EXMS2	FRCC	INTM	ISPR
	MD0	MD0H	MD0L	MD1	MD1H	MD1L	MM
	P0	P0H	P0L	P1	P2	P3	P4
	P5	PM0	PM1	PM2	PM3	PM5	PMC2
	PMC3	PSW	PSWH	PSWL	PWM0	PWM0H	PWM0L
	PWM1	PWM1H	PWM1L	PWMM	RFM	RTPC	RXB
	SCC	SCM	SEIC	SP	SPH	SPL	SRIC
	SRMS	STBC	STIC	STMS	TBIC	TBM	TM0
	TM0H	TM0L	TM1	TM1H	TM1L	TMC0	TMC1
	TMIC0	TMIC1	TMIC2	TMMS0	TMMS1	TMMS2	TXB
	UDC0	UDC0H	UDC0L	UDC1	UDC1H	UDC1L	UDCC0
	UDCC1	WDM					
	µPD78320/µPD78322						
	ISPR	ADCR	ADCRH	ADM	ASIM	ASIS	BRG
	BRGM	CC01LW	CC01UW	CCW	CCX0UW	CCX0LW	CM00
	CM01	CM02	CM03	CM10	CM11	CSE0	CSE0H
	CSE0L	CSE1	CSE1L	CSIM	CT01LW	CT01UW	CT02LW
	CT02UW	CT03LW	CT03UW	CTX0LW	CTX0UW	FCC	IF0
	IF0H	IF0L	IF1	IF1H	INTM0	INTM1	ISM0
	ISM0H	ISM0L	ISM1	ISM1L	MK0	MK0H	MK0L
	MK1	MK1L	MM	P0	P2	P3	P4
	P5	P7	P8	P9	PB0	PB0H	PB0L
	PB1	PB1L	PM0	PM3	PM5	PM8	PM9
	PMC0	PMC3	PMC8	PRDC	PRM	PRSL	PWC
	RPUM	RTP	RTPR	RTPS	RXB	SBIC	SIO
	STBC	TM0LW	TM0UW	TM1	TMC	TOC0	TOC1
	TXS	WDM					

sfr symbols	μPD78327/μPD78328						
	ADCR	ADCRH	ADM	ASIM	ASIS	BRG	BRGM
	CC10	CCW	CM00R	CM00S	CM01R	CM01S	CM02R
	CM02S	CM03R	CM03S	CM04R	CM04S	CM05R	CM05S
	CM06	CM20	CSE0	CSE0H	CSE0L	CSE1	CSE1L
	CSIM	FCC	IF0	IF0H	IF0L	IF1	IF1L
	INTM0	ISM0	ISM0H	ISM0L	ISM1	ISM1L	ISPR
	MK0	MK0H	MK0L	MK1	MK1L	MM	P0
	P0L	P2	P3	P4	P5	P7	P8
	P9	PB0	PB0H	PB0L	PB1	PB1L	PM0
	PM3	PM5	PM8	PM9	PMC3	PMC8	P0H
	PRDC	PRSL	PWC	PWMB	PWMC	RTPC	RXB
	SBIC	SIO	STBC	TM0	TM1	TM2	TMC0
	TMC1	TOUT	TUM	TXS	WDM		
	μPD78330/μPD78334						
	ADCR0	ADCR0H	ADCR1	ADCR1H	ADCR2	ADCR2H	ADCR3
	ADCR3H	ADCR4	ADCR4H	ADCR5	ADCR5H	ADCR6	ADCR6H
	ADCR7	ADCR7H	ADM	ASIM	ASIS	BRG	BRGM
	CC00R	CC01R	CCW	CM01R	CM02R	CM03R	CM04R
	CM11	CM12	CM20	CM21	CM30	CMX0	CSE0
	CSE0H	CSE0L	CSE1	CSE1L	CSIM	CT00	CT01
	CT02	CT10	FCC	IF0	IF0H	IF0L	IF1
	IF1L	INTM0	INTM1	ISM0	ISM0H	ISM0L	ISM1
	ISM1L	ISPR	MK0	MK0H	MK0L	MK1	MK1L
	MM	P0	P1	P2	P3	P4	P5
	P7	P8	P9	PB0	PB0H	PB0L	PB1
	PB1L	PM0	PM1	PM3	PM5	PM9	PMC0
	PMC1	PMC3	PPOS	PRDC	PRSL	PWC	PWM0
	PWM1	PWMC	RTP	RTPR	RTPS	RXB	SBIC
	SETM	SFTM	SIO	STBC	TLA	TM0	TM1
	TM2	TM3	TMC0	TMC1	TOC0	TOC1	TUM0
	TUM1	TXS	WDM				
Registers	A	AX	B	BC	C	CY	DE
	E	H	HL	L	R0	R1	R2
	R3	R4	R5	R6	R7	R8	R9
	R10	R11	R12	R13	R14	R15	RP0
	RP1	RP2	RP3	RP4	RP5	RP6	RP7
	RB0	RB1	RB2	RB3	RB4	RB5	RB6
	RB7	UP	UPH	UPL	VP	VPH	VPL
	X						

Seg. attr.	AT	CALLTO	CALLT1	FIXED	SADDR	SADDRP	UNIT

*Seg. attr.: Segment attributes

A.4 List of Reserved Words for 78K/VI

Instructions	ADDB	ADDCB	ADDCD	ADDCW	ADDD	ADDW	ADJBA
	ADJBS	AND1B	AND1W	ANDB	ANDD	ANDW	BCS
	BES	BFBS	BFSETBS	BFSETWS	BFWS	BGES	BGTS
	BHS	BIG_SEM	BLES	BLS	BLTS	BNCS	BNES
	BNHS	BNLS	BNS	BNVS	BNZS	BPES	BPOS
	BPS	BR	BR1Z	BRK	BRKCS	BRKT	BRM
	BRS	BTBS	BTCLRBS	BTCLRWS	BTWS	BVS	BZS
	CALL	CALLT	CHKL	CHKLR	CLR1	CLR1B	CLR1W
	CMPB	CMPBKCB	CMPBKCW	CMPBKEB	CMPBKEW	CMPBKNCB	CMPBKNCW
	CMPBKNEB	CMPBKNEW	CMPD	CMPMCB	CMPMCW	CMPMEB	CMPMEW
	CMPMNCB	CMPMNCW	CMPMNEB	CMPMNEW	CMPW	CVTBW	CVTWD
	DBNZ	DBNZE	DBNZNE	DECB	DECW	DI	DIVD
	DIVUD	DIVUM	DIVW	DSBNZ	EI	INCB	INCW
	INIT	IRSM_TSK	MACW	MOV1B	MOV1W	MOVB	MOVBKB
	MOVBKW	MOVD	MOVEA	MOVMD	MOVMW	MULB	MULUB
	MULUW	MULW	NEGB	NEGW	NOP	NOT1	NOT1B
	NOT1W	NOTB	NOTW	NOVW	OR1B	OR1W	ORB
	ORD	ORW	POP	POPR	POPU	PREQ_SEM	PUSH
	PUSHR	PUSHU	QH_OUT	QS_OUT	QT_IN	RET	RETB
	RETCS	RETCSB	RETI	RET_INT	RET_RSM	ROL4	ROLB
	ROLCB	ROLCD	ROLCW	ROLD	ROLW	ROR4	RORB
	RORCB	RORCD	RORCW	RORD	RORW	RSM_TSK	SEL
	SET1	SET1B	SET1W	SHLB	SHLD	SHLW	SHRAB
	SHRAD	SHRAW	SHRB	SHRD	SHRW	SUBB	SUBCB
	SUBCD	SUBCW	SUBD	SUBW	SUS_TSK	WAI_SEM	XCHB
	XCHBKB	XCHBKW	XCHMB	XCHMW	XCHW	XOR1B	XOR1W
	XORB	XORD	XORW				
Operators	AND	EQ	GE	GT	HIGH	LE	LOW
	LT	MOD	NE	NOT	OR	SHL	SHR
	XOR						
Directives	BR	BSEG	CSEG	DB	DBIT	DS	DSEG
	DW	END	ENDM	ENDS	EQU	EQUD	EXITM
	EXTBIT	EXTRN	IRP	LOCAL	MACRO	NAME	ORG
	PUBLIC	REPT	SET				

Control instructions	COND	DEBUG	DG	EJ	EJECT	ELSE	ELSEIF
	_ELSEIF	ENDIF	GEN	IC	IF	_IF	INCLUDE
	LI	LIST	NOCOND	NODEBUG	NODG	NOGEN	NOLI
	NOLIST	NOXR	NOXREF	PC	PROCESSOR	RESET	SET
	SUBTITLE	ST	TITLE	TT	XR	XREF	
sfr symbols	ADCR0	ADCR1	ADCR2	ADCR3	ADCR4	ADCR5	ADCR6
	ADCR7	ADCRH0	ADCRH1	ADCRH2	ADCRH3	ADCRH4	ADCRH5
	ADCRH6	ADCRH7	ADM	ASIM	ASIS	BRG	BSC
	CC00LW	CC00UW	CC01LW	CC01UW	CC02LW	CC02UW	CC03LW
	CC03UW	CC10	CC11	CM00	CM01	CM02	CM03
	CM10	CM11	CM20	CM21	CRC	CSE0	CSE0H
	CSE0L	CSE1L	CSIM	CT20	CT21	IF0	IF0H
	IF0L	IF1L	INTM0	INTM1	IPGC0	IPGC1	IPGCM0
	IPGCM1	IPGCM2	IPGCM3	IPGCM4	IPGCM5	IPGCM6	IPGCM7
	IPGCMS	IPGCT	IPGS0	IPGS1	IPGTM	ISM0	ISM0H
	ISM0L	ISM1L	ISPR	MK0	MK0H	MK0L	MK1L
	MM	OTC	P0	P1	P2	P3	P4
	P5	P7	P8	P9	PB0	PB0H	PB0L
	PB1L	PM0	PM1	PM3	PM5	PM8	PM9
	PMC0	PMC3	PMC8	PRM0	PRM1	PRSL	PWC
	RPDC	RPUM	RTP	RTPR	RTPS	RxB	SBIC
	SIO	STBC	TM0LW	TM0UW	TM1	TM2	TMC0
	TMC1	TOC0	TOC1	TxS	WDM		
Registers	CY	R0	R1	R2	R3	R4	R5
	R6	R7	R0H	R0L	R1H	R1L	R2H
	R2L	R3H	R3L	R4H	R4L	R5H	R5L
	R6H	R6L	R7H	R7L	RB0	RB1	RB2
	RB3	RP0	RP1	RP2	RP3	SP	
Seg. attr.	AT	DSADDR	SADDR	TABLE	UNIT	WSADDR	

*Seg. attr.: Segment attributes

APPENDIX B. LIST OF DIRECTIVES

Table B-1. List of Directives

No.	Directive				Function/ classification
	Symbol field	Mnemonic field	Operand field	Comment field	
1	[segment name]	CSEG	[reloc. attr.]	[;comment]	Declares the start of a code segment.
2	[segment name]	DSEG	[reloc. attr.]	[;comment]	Declares the start of a data segment.
3	[segment name]	BSEG	[reloc. attr.]	[;comment]	Declares the start of a bit segment.
4	[segment name]	ORG	absol. expres.	[;comment]	Declares the start of an absolute segment. (See Note 1.)
5	None	ENDS	None	[;comment]	Indicates the end of the segment.
6	name	EQU	expression	[;comment]	Defines a name. (See Note 2.) name:symbol
7	name	EQUD	32-bit imm. data/ symbol	[;comment]	Defines a name. (See Note 2.) name: symbol (for 78K/VI only)
8	name	SET	absol. expres.	[;comment]	Defines a relocatable name. (See Note 1.) name:symbol
9	[label:]	DB	{(size) [initial value [,...]}	[;comment]	Initializes or reserves a byte data area. (See Note 3.) label:symbol

Table B-1. List of Directives (contd)

No.	Directive				Function/ classification
	Symbol field	Mnemonic field	Operand field	Comment field	
10	[label:]	DW	(size) [initial value [,...]	[;comment]	Initializes or reserves a word data area. label:symbol
11	[label:]	DS	absol. expres.	[;comment]	Reserves a byte data area. (See Note 1.) label:symbol
12	[name]	DBIT	None	[;comment]	Reserves a bit data area. (See Note 1.) name:symbol
13	[label:]	PUBLIC	symbol name [,...]	[;comment]	Declares an external definition name.
14	[label:]	EXTRN	symbol name [,...]	[;comment]	Declares an external reference name.
15	[label:]	EXTBIT	bit symbol name [,...]	[;comment]	Declares an external reference name. Symbol names are limited to those having a bit value.
16	[label:]	NAME	object module name [,...]	[;comment]	Defines a module name. module name:symbol
17	[label:]	BR	expression [,...]	[;comment]	Automatically selects a Branch instruction. label:symbol

Table B-1. List of Directives (contd)

No.	Directive				Function/ classification
	Symbol field	Mnemonic field	Operand field	Comment field	
18	[label:]	RSS	n	[;comment]	Declares the value of the Register Set Select flag. n=0, 1 (for 78K/III only)
19	name	MACRO	[formal parameter [,...]]	[;comment]	Defines a macro. macro name:symbol
20	[label:]	LOCAL	symbol name [,...]	[;comment]	Defines a symbol valid only within the macro. (See Note 4.) label: symbol
21	[label:]	REPT	absol. expres.	[;comment]	Defines the repeat count in macro-expansion. label: symbol
22	[label:]	IRP	formal parameter, <actual parameter [,...]>	[;comment]	Expands the macro body by replacing formal parameters with actual parameters. label:symbol
23	[label:]	EXITM	None	[;comment]	Interrupts the macroexpansion. (See Note 4.)
24	None	ENDM	None	[;comment]	Indicates the end of macrodefinition. (See Note 4.)
25	None	END	None	[;comment]	Indicates the end of the source module

Notes: 1. Forward reference of a symbol is not allowed in the expression described in the Operand field.

2. Neither forward reference of a symbol nor reference of an external reference name is allowed in the expression described in the Operand field.

3. A character string may be described in place of an initial value.

4. This directive can be used only in the macro-definition.

APPENDIX C. MAXIMUM PERFORMANCE CHARACTERISTICS

(1) Maximum performance characteristics of Assembler

Item		Restriction
Symbol length	w/o -S option	8 characters
	with -S option	31 characters
No. of characters per line		130 characters
No. of segments		100 segments

(2) Maximum performance characteristics of Linker

Item	Limit
Number of input modules files	64 files

(3) Restrictions on number of symbols

	No. of local symbols	No. of PUBLIC symbols
Assembler	Approx. 2,900 symbols	(see Note 1)
Linker	2,900 symbols x No. of modules	Approx. 3,000 symbols (see Note 2)

NOTE: 1. There is no restriction on the number of symbols by symbol type.

2. If the number of PUBLIC symbols exceeds 1,000, the execution speed slows down because of the additional time required to access a temporary file.

NEC