

To our customers,

---

## Old Company Name in Catalogs and Other Documents

---

On April 1<sup>st</sup>, 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<sup>st</sup>, 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.



To all our customers

---

## **Regarding the change of names mentioned in the document, such as Mitsubishi Electric and Mitsubishi XX, to Renesas Technology Corp.**

---

The semiconductor operations of Hitachi and Mitsubishi Electric were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Mitsubishi Electric, Mitsubishi Electric Corporation, Mitsubishi Semiconductors, and other Mitsubishi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Note : Mitsubishi Electric will continue the business operations of high frequency & optical devices and power devices.

Renesas Technology Corp.  
Customer Support Dept.  
April 1, 2003

# RASM77 V.5.10

User's Manual

Relocatable Assembler for 77xx Series

- Microsoft, MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the U.S. and other countries.
- Sun, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. or other countries, and are used under license.
- Linux is a trademark of Linus Torvalds.
- Turbolinux and its logo are trademarks of Turbolinux, Inc.
- IBM and AT are registered trademarks of International Business Machines Corporation.
- Intel and Pentium are registered trademarks of Intel Corporation.
- Adobe, Acrobat, and Acrobat Reader are trademarks of Adobe Systems Incorporated.
- All other brand and product names are trademarks, registered trademarks or service marks of their respective holders.

**Keep safety first in your circuit designs!**

- Renesas Technology Corporation and Renesas Solutions Corporation put the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

**Notes regarding these materials**

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation, Renesas Solutions Corporation or a third party.
- Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation and Renesas Solutions Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor for the latest product information before purchasing a product listed herein. The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors. Please also pay attention to information published by Renesas Technology Corporation and Renesas Solutions Corporation by various means, including the Renesas home page (<http://www.renesas.com>).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation and Renesas Solutions Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination. Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation or Renesas Solutions Corporation for further details on these materials or the products contained therein.

For inquiries about the contents of this document or product, fill in the text file the installer generates in the following directory and email to your local distributor.

¥SUPPORT¥Product-name¥SUPPORT.TXT

Renesas Tools Homepage <http://www.renesas.com/en/tools>

---

# Preface

RASM77 is a relocatable assembler for the 7700 Family microcomputers. RASM77 creates machine language data files, debugging information files, etc. from source programs written in 7700 Family assembly language for the 7700 Family microcomputers. This user's manual describes the functions and operation of the following programs that make up the software product RASM77:

1. Relocatable assembler RASM77
2. Structured preprocessor PRE77
3. Linkage editor LINK77
4. Librarian LIB77
5. Cross-referencer CRF77

This manual does not guarantee nor authorize the right to use the software.

## Organization of RASM77 User's Manual

The RASM77 User's Manual consists of five parts as described below. Each part describes each of the five programs that make up RASM77 Assembler in the same sequence as much as possible. For example, explanation of environment variables for each program can be found in the chapter that describes operation of that program.

- Part 1: RASM77 Operation Manual  
Describes the method of using the relocatable assembler program RASM77 and the method of coding source programs.
- Part 2: PRE77 Operation Manual  
Describes method of using preprocessor language.
- Part 3: LINK77 Operation Manual  
Describes method of executing the linker program LINK77 and the functions of its sections.
- Part 4: LIB77 Operation Manual  
Describes the method of using the librarian program LIB77.
- Part 5: CRF77 Operation Manual  
Describes the method of using the cross-referencer program CRF77.

PART 1

RELOCATABLE MACRO ASSEMBLER  
FOR 7700 FAMILY



# RASM77 OPERATION MANUAL

---

# Table of Contents

## Chapter 1. RASM77 User's Manual Organization

## Chapter 2. Overview

2.1 Functions .....	3
2.2 Files Created by RASM77 .....	4
2.3 Structure of PRN File .....	4
2.4 Structure of TAG File .....	12

## Chapter 3. Source Program Coding Method

3.1 Structure of Source Program .....	13
3.2 Line Formats .....	14
3.2.1 Instruction Line .....	14
3.2.2 Structured Preprocessor Instruction Line .....	14
3.2.3 Pseudo Instruction Line .....	14
3.2.4 Macro Instruction Line .....	15
3.2.5 Comment Line .....	15
3.3 Field Coding Method .....	15
3.3.1 Symbol/Label Field .....	15
3.3.2 Op-code/Pseudo Instruction Field .....	16
3.3.3 Operand Field .....	16
3.3.4 Comment Field .....	16
3.4 Operand Field Coding Method .....	16
3.4.1 Data Format .....	16
3.4.2 Instructions .....	17

## Chapter 4. Instruction Coding Method

4.1 Addressing Mode .....	19
4.2 Data Length Specification .....	21
4.3 Setting Direct Page and Absolute Addressing .....	23
4.4 Addressing Mode Selection .....	23
4.4.1 Setting the Direct Page Register and Data Bank Register .....	24
4.4.2 Addressing Mode During Symbol•Absolute Value Operation .....	25
4.4.3 Addressing Mode During Label Operation .....	26
4.4.4 Disabling Addressing Mode Selection .....	28



---

## Chapter 5. Pseudo Instruction Coding Method

<b>5.1 Function of Pseudo Instructions .....</b>	<b>30</b>
<b>5.2 Assembly Control Pseudo Instructions .....</b>	<b>32</b>
5.2.1 Data Length Declaration .....	32
5.2.2 DPR and DT Value Declaration .....	32
5.2.3 Conditional Assembly .....	32
5.2.4 Include File .....	32
5.2.5 Equation .....	32
5.2.6 Declare End of Assembly .....	32
5.2.7 Message Output .....	33
5.2.8 Assembly Error Output .....	33
5.2.9 Define String .....	33
<b>5.3 Address Control Pseudo Instructions .....</b>	<b>33</b>
5.3.1 Address Declaration .....	33
5.3.2 Memory Allocation .....	33
5.3.3 Data Definition .....	33
5.3.4 Correct Address Alignment .....	33
<b>5.4 Linkage Control Pseudo Instructions .....</b>	<b>34</b>
5.4.1 Section Name Specification .....	34
5.4.2 Global Label Name Specification .....	34
5.4.3 Linkage Filename Specification .....	34
5.4.4 Version Control .....	34
<b>5.5 Listing Control Pseudo Instructions .....</b>	<b>34</b>
<b>5.6 Source Level Debug Support .....</b>	<b>35</b>
<b>5.7 Reserved Pseudo Instructions .....</b>	<b>35</b>

## Chapter 6. Macro Instruction

<b>6.1 Macro Instruction Functions .....</b>	<b>36</b>
<b>6.2 Macro Instruction Types .....</b>	<b>36</b>
<b>6.3 Macro Operators .....</b>	<b>37</b>

## Chapter 7. Operation

<b>7.1 Starting RASM77 .....</b>	<b>41</b>
<b>7.2 Input Parameters .....</b>	<b>41</b>
7.2.1 Source Filename .....	41
7.2.2 Command Parameters .....	41
<b>7.3 Input Method .....</b>	<b>44</b>
<b>7.4 Errors .....</b>	<b>47</b>
7.4.1 Error Types .....	47
7.4.2 Return Values to MS-DOS .....	49
<b>7.5 Environment variables .....</b>	<b>49</b>

---

## Appendix A. Error Messages

A.1 System Error Messages .....	50
A.2 Assembly Error Messages .....	52
A.3 Warning Messages .....	57

## Appendix B. Pseudo Instructions

B.1 Conventions .....	59
B.2 Pseudo Instructions .....	59
B.3 Debugging Pseudo Instructions .....	83
B.4 Reserved Pseudo Instructions .....	87

## Appendix C. Macro Instructions

C.1 Conventions .....	93
C.2 Macro Instructions .....	93

## Appendix D. Instruction Set

D.1 Symbols .....	105
D.2 Instruction Set .....	107

## Appendix E. Instruction by Addressing Mode

E.1 Instruction by Addressing Mode .....	121
E.2 Addressing Mode Relationship Table .....	128
E.3 Selection of Addressing Mode .....	129

---

## List of Figures

Figure 2.1 PRN File Example (Beginning of Source File) .....	6
Figure 2.2 PRN File Example (Middle of Source File) .....	7
Figure 2.3 PRN File Example (End of Source File) .....	8
Figure 2.4 PRN File Example (Assembly Information) .....	8
Figure 2.5 PRN File Example (Symbol and Label List) .....	9
Figure 2.6 PRN File Example (Macro and Include Expansion) .....	10
Figure 2.7 PRN File Example (Structured Preprocessor Section) .....	11
Figure 2.8 TAG File Example .....	12
Figure 4.1 Example of conditional assemble based on data and index lengths .....	22
Figure 7.1 Example of RASM77 Startup Command Line .....	44
Figure 7.2 HELP Screen for Command Line Error .....	45
Figure 7.3 Normal Termination Screen .....	46
Figure 7.4 Error Display Example .....	48

---

## List of Tables

Table 3.1 List of Operators .....	18
Table 4.1 Association of CPU Internal Flags and Assembler .....	21
Table 4.2 Function of M_FLAG and X_FLAG .....	22
Table 4.3 Relationship between DPR, DT and Assembler .....	23
Table 6.1 List of Macro Instructions .....	38
Table 7.1 List of Command Parameters .....	42
Table 7.2 Listing of Error Levels .....	49
Table A.1 List of System Error Messages .....	51
Table A.2 List of Assembly Errors .....	52
Table A.3 List of Warning Messages .....	58
Table B.1 Allowed Logical Instructions .....	72
Table D.1 Symbols for Instruction List.....	106
Table D.2 Instructions .....	108
Table E.1 Addressing Mode Table .....	128

---

# CHAPTER 1

## RASM77 User's Manual Organization

The RASM77 Operation Manual consists of the following chapters:

- Chapter 2. Overview  
Describes the basic functions of the RASM77 relocatable assembler and the files created by RASM77.
- Chapter 3. Source Program Coding Method  
Describes the structure of assembly language source programs that are processed by RASM77.
- Chapter 4. Instruction Coding Method  
Explains how to code 7700 Family instructions that can be used by RASM77.
- Chapter 5. Pseudo Instruction Coding Method  
Describes functions of and explains how to code the pseudo instructions provided by RASM77.
- Chapter 6. Macro Instruction  
Describes the macro instructions available with RASM77.
- Chapter 7. Operation  
Explains how to input RASM77 commands.
- Appendix A. Error Messages  
Lists error messages output by RASM77 along with explanation of the errors and actions to be taken.
- Appendix B. Pseudo Instructions  
Lists and explains all pseudo instructions provided by RASM77.
- Appendix C. Macro Instructions  
Lists and explains all pseudo instructions provided by RASM77.
- Appendix D. Instruction Set  
Lists all 7700 Family instructions provided by RASM77.

## CHAPTER 1. RASM77 USER'S MANUAL ORGANIZATION

---

- Appendix E. Instruction Sets by Addressing Modes  
Lists the 7700 Family instructions provided by RASM77 in each addressing mode.

# CHAPTER 2

## Overview

RASM77 converts a source program written in assembly language (hereafter referred to as a source file) into a relocatable file that can be processed by LINK77<sup>1</sup> and LIB77<sup>2</sup>. This step is referred to as assembly. Relocatable files are converted into machine language data by LINK77.

### 2.1 Functions

Development of a large software requires functions that enable several engineers to share programming resources such as data and existing codes. RASM77 offers the following functions to facilitate this task.

1. The user can specify any section name desired by the pseudo instruction, `.SECTION`.
2. Because there is no limit to the number of sections that can be specified, address specification is possible for linkage editing even on a user system that has ROM and RAM memories that are divided into large number of separate areas.
3. Versions of the relocatable files being linked can be verified by specifying the version number with the pseudo instruction `.VER`.
4. The relocatable files to be linked can be specified by the pseudo instructions `.LIB` and `.OBJ`. (This feature eliminates the need for specifying filenames during linkage editing.)

RASM77 also has the following assembly functions:

1. The most efficient addressing mode is automatically selected based on the values in the direct page register (DPR) and data bank register (DT).
2. A TAG<sup>3</sup> file that stores error information can be created. (This feature enables efficient assembly error correction.)
3. Because RAM and ROM areas can coexist in a file, RASM77 can also be used as an

absolute assembler. (Linkage is required.)

---

<sup>1</sup> LINK77 is the program name of the Series 7700 Family linkage editor.

<sup>2</sup> LIB77 is the program name of the Series 7700 Family librarian.

<sup>3</sup> This file is called a TAG file because it contains "tags" that show the locations of errors and warnings.

### 2.2 Files Created by RASM77

RASM77 creates three types of files as described below.

1. Relocatable file
  - A relocatable file contains machine language data and its relocation information.
  - A relocatable file contains symbol information for use in symbolic debugging.
  - A relocatable file can be linked by LINK77 to create Intel HEX format machine language data.
  - Relocatable file is not created if an assembler error has occurred.
  - File extension of relocatable file is .R77.
  - Relocatable files should not be output to a printer or screen because they are in binary format.
  
2. Print file (hereafter referred to as PRN file)
  - A PRN file contains source file data, addresses of source file data locations and various data created by RASM77.
  - A PRN file can be printed and used for debugging.
  - A PRN file is generated when the command parameter “-L” is specified.
  - The file extension of PRN files is .PRN.
  - The structure of PRN files is described in detail in Section 2.3.
  
3. TAG file
  - A TAG file contains the assembly error messages and warning messages that were generated during assembly.
  - The file extension of TAG files is .TAG.
  - TAG file should be referred to when making error corrections using an editor.
  - A TAG file is generated when the command parameter “-E” is specified.
  - The structure of TAG files is described in detail in Section 2.4.

### 2.3 Structure of PRN File

Figures 2.1 through 2.5 show sample PRN files. A PRN file contains the following information:

1. Source file data, addresses of source file data locations and data created by RASM77 (Figures 2.1 to 2.3 show this portion of PRN file.)
  - A line that references an external label<sup>4</sup> is marked by ‘E’ next to the source file data.
  - A line that references a public label<sup>5</sup> is marked by ‘P’ next to the source file data.
  - A line that references a local label<sup>6</sup> is marked by ‘L’ next to the source file data.



- A line that is an expansion of a macro is marked by '+' next to the source file data.
  - A line that references a symbol defined with .DEFINE is marked by '-' next to the source file data.
  - The .INCLUDE nesting level is marked by the number corresponding to the level next to the source file data (Figure 2.6 shows this portion of PRN file).
  - Structured description source lines are output to a print file as comment lines. (Figure 2.7 shows this portion of PRN file).
2. Assembly information (Figure 2.4 shows this portion of PRN file)  
This portion of a PRN file shows the number of errors, number of warnings, total number of lines, number of comment lines and the memory size of each section.
  3. Symbols listing (Upper section of Figure 2.5 shows this portion of PRN file)  
The number of symbols per line in the symbol list depends on the ".COL" pseudo instruction. If the number of columns specified with ".COL" is 99 or less, the number of symbols output is 4, if it is between 100 and 119 columns, the number of symbols output is 5, if it is 120 or more, the number of symbols output is 6.

This portion of a PRN file lists the symbols and their values as they are defined in the program in three groups:

- -D OPTION  
Each symbol in this group is defined by specifying the command parameter "- D" in the command line and referenced within the program.
  - EQUATE  
Each symbol in this group is defined by the pseudo instruction .EQU and referenced within the program.
  - UNUSED  
Each symbol in this group is defined by either the command parameter "-D" or .EQU but not referenced in the program.
4. Labels list (Lower section of Figure 2.5 shows this portion of PRN file)  
This portion of a PRN file lists the labels and their values as they are defined in the program in two groups:
    - USED  
Each label in this group is defined and referenced in the program.
    - UNUSED  
Each label in this group is defined but not referenced in the program.
  5. When the number of columns specified by the pseudo instruction .COL is 132, the assembly execution date and time is placed in the header of the PRN file list in the following format:

Sun Mar 31 15:06:42 1991

---

<sup>4</sup> Refers to a label that is defined in another file. External labels and public labels are referred to as global labels.

<sup>5</sup> Refers to a label that is defined in this file and can be referenced by other files.

<sup>6</sup> Refers to a label that is defined in this file and can be referenced only within the file.

```
1           ;
2           ;      Start_up Routine
3           ;
4
5           ;
6           ;      Data section
7           ;      Initialized static variable area
8           ;
9           .SECTION   DATA
10          .PUB       DATATOP
11 000000          DATATOP:
12 (000000)    8H BYTE  NULLDT: .BLKB    8
13 (000008)   1AH BYTE  TITLE:  .BLKB    26
14
15          ;
16          ;      BSS section
17          ;      Uninitialized static variable area
18          ;
19          .SECTION   BSS
20          .PUB       BSSTOP
21 000000          BSSTOP:
22 (000000)    1H BYTE  WORK1:  .BLKB    1
23 (000001)    1H BYTE  WORK2:  .BLKB    1
24 (000002)    2H BYTE  WORK3:  .BLKW    1
25
26          ;
27          ;      HEAP section
28          ;      Area used by memory handling functions such as malloc
29          ;
30          .SECTION   HEAP
31          .PUB       HEAPTOP
32 000000          HEAPTOP:
33 (000000)  1000H BYTE  HEAP_A: .BLKB    1000H
34
35          ;
36          ;      STACK section
37          ;      stack area
38          ;
39          .SECTION   STACK
40          .PUB       STKTOP
41 (000000)  1000H BYTE  .BLKB    1000H
42 001000          STKTOP:
```

**Figure 2.1 PRN File Example (Beginning of Source File)**

```

43          .PAGE
44          ;
45          ; PROGRAM section
46          ; Program area
47          ;
48          .SECTION PROGRAM
49          .EXT MAIN
50          .DPEXT DPPAGE1:WK1,WK2
51          .DTEXT DTPAGE1:TBL1,TBL2
52          .PUB _INIT
53          .DATA 16
54          .INDEX 16
55          E .DP OFFSET DPPAGE1
          ↑ Indicates external label reference
56          E .DT BANK DTPAGE1
57          _INIT:
58          000000 C2FF CLP #0FFH
59          000002 A90010 P LDA A,#STKTOP
          ↑ Indicates public label reference
60          000005 1B TAS
61          000006 A90000 LDA A,#SIZEOF DATA
62          000009 A20000 P LDX #OFFSET CONSTOP
63          00000C A00000 P LDY #OFFSET DATATOP

```

Figure 2.2 PRN File Example (Middle of Source File)

## CHAPTER 2. OVERVIEW

```

64 00000F 540000      P      MVN      BANK CONSTOP, BANK DATATOP
65 000012 AD0000      E      LDA      A,DT:TBL1
66 000015 *42AD0000  E      LDA      B,TBL2
      ↑ Indicates optimization
67 000019 A20000      E      LDX      #WK1
68 00001C 202200      L      JSR      _SUB
      ↑ References a local label
69 00001F 4C0000      E      JMP      MAIN
70
71 000022          ;
72 000022 9500          _SUB:    STA  A,0,X
73 000024 42950A          STA  B,10,X
74 000027 60          RTS
75
76          ;
77          ;      CONST section
78          ;      Initialized data area
79          ;
80          .SECTION  CONST
81          .PUB      CONSTOP
82 000000
83 000000 000000000000    .BYTE  0,0,0,0,0,0,0,0
      000006 0000
84 000008 4D3337373030 .BYTE  'M37700 C Compiler Ver 1.0',0
      00000E 204320436F6D
      000014 70696C657220
      00001A 56657220312E
      000020 3000
85          ;
86          .END

```

Figure 2.3 PRN File Example (End of Source File)

```

ERROR  COUNT  00000  (0000H)
WARNING COUNT 00000  (0000H)
TOTAL   LINE  00086  (0056H) LINES
COMMENT LINE  00033  (0021H) LINES
DATA          00000034 (000022H) BYTES
↑ Indicates the section name
BSS          00000004 (000004H) BYTES
HEAP         00004096 (001000H) BYTES
STACK        00004096 (001000H) BYTES
PROGRAM      00000040 (000028H) BYTES
CONST        00000034 (000022H) BYTES

```

Figure 2.4 PRN File Example (Assembly Information)

```

*** SYMBOLS ( TYPE = -d OPTION ) ***

*** SYMBOLS ( TYPE = EQUATE ) ***

*** SYMBOLS ( TYPE = UNUSED ) ***

*** LABELS ( TYPE = USED ) ***

_SUB      000022' CONSTOP      000000p DATATOP      000000p DPPAGE1      000000e
          ↑ Indicates a local label

DTPAGE1   000000e MAIN        000000e STKTOP      001000p TBL1        000000e
          ↑ Indicates a public label

TBL2      000000e WK1         000000e
          ↑ Indicates an external label

*** LABELS ( TYPE = UNUSED ) ***

_INIT     000000p BSSTOP      000000p HEAP_A       000000' HEAPTOP      000000p
NULLDT    000000' TITLE      000008' WK2         000000e WORK1       000000'
WORK2     000001' WORK3      000002'

```

Figure 2.5 PRN File Example (Symbol and Label List)

## CHAPTER 2. OVERVIEW

```

31 000014          INITIAL:
32                .section      prog
33                .include      initial.a77
34                1              .DATA      16
35                1              .INDEX     16
36                1              .DP        0
37                1              .DT        0
38 000000 78       1              SEI
39 000001 C238     1              CLP        m,x,D
40                1
41                .include      clr_ram.a77; Call in
                                   ; initial.a77

42 000003 A27F02   2              LDX      #027FH
43 000006 9A       2              TXS
44 000007 A90000   2              LDA      A,#0
45 00000A 5B       2              TAD
46 00000B 89C200   2              LDT      #0
47 00000E A90000   2              LDA      A,#0
48 000011          2 RAM_CLEAR:
49 000011 *9500    2              STA      A,0,X
50 000013 CA       2              DEX
51 000014 CA       2              DEX
52 000015 E07E00   2              CPX      #07EH
53 000018 D0F7     L2          BNE      RAM_CLEAR
54                2
55                1              .DATA      8
56 00001A F8       1              SEM
57 00001B 58       1              CLI
                    ↑
                    Indicates .INCLUDE nesting level

58 00001C          MAIN:
59 00001C A00A00    1              LDY      #10
60 00001F          LOOP:
61 00001F          1              ADD      A,WORK,Y      ; Macro call
62 00001F 18       +              CLC
63                +              .IF      "Y"
64 000020 790000   L +          ADC      A,WORK,Y
65                +              .ELSE
66                +              .ENDIF
67                +              .ENDM
68                +
                    ↑
                    Indicates macro expansion

69 000023 990A00   L              STA      A,DATA,Y
70 000026 88       1              DEY
71 000027 C00000   1              CPY      #0
72 00002A D0F3     L              BNE      LOOP
73                1              .END

```

Figure 2.6 PRN File Example (Macro and Include Expansion)

```

1          ; *** 7700 Family PREPROCESSOR V5.00.00 ***
2          .language      PRE77_Rev01
3          .source        sample.p77
4          .SECTION      RAM
5          .func          _sample_0
6          .ORG          0000H
7 (000000)      1H BYTE   WORK1: .BLKB   1
8 (000001)      1H BYTE   WORK2: .BLKB   1
9
10         .endfunc      _sample_0
11        .SECTION      INIT
12        .func          _sample_1
13        ;FLAG_0 .EQU    0,WORK1← Structured code source line
14        FLAG_0 .define  WORK1← Structured code expansion
15        ;FLAG_1 .EQU    1,WORK1
16        FLAG_1 .define  WORK1
17        .endfunc      _sample_1
18        .section      structprog
19        .func          _sample_2
20        ; for [ DP:FLAG_0 ] == 0
21        .cline      9 ← Source level debugging information
22        000000      ..F1: ← Structured code label
23        000000 2400010011 -L      BBS      #00001H,DP:WORK1,..F2
24        ↑ Indicates reference to string definition symbol
25        ; if [ DP:FLAG_1 ] == 1
26        .cline      10
27        000005 340002000A -L      BBC      #00002H,DP:WORK1,..I3
28        ; [ WORK1 ] = 0
29        .cline      11
30        00000A A90000      LDA      A,#0
31        00000D *8500      L        STA      A,WORK1
32        ; [ WORK2 ] = 0
33        .cline      12
34        00000F A90000      LDA      A,#0
35        000012 *8501      L        STA      A,WORK2
36        ; endif
37        .cline      13
38        000014      ..I3:
39        000014 80EA      L        BRA      ..F1
40        ; next
41        .cline      14
42        000016      ..F2:
43        .endfunc      _sample_2
44        .end

```

Figure 2.7 PRN File Example (Structured Preprocessor Section)

### 2.4 Structure of TAG File

Figure 2.6 shows a sample of a TAG file. A TAG file contains the following information:

1. Source information  
For each occurrence of error or warning, list line number, location, object code and source file contents are specified.
2. TAG information  
For each occurrence of error or warning identified by source information, filename, line number within the file, sequential line number, error number and error message are specified.

The TAG file should be printed and referenced when correcting errors with an editor.

```
115 00F025 EAEA          BCC    LOOP2
TEST.ASM 115 ( TOTAL LINE 115 ) Error 18: Relative jump is out of range
127 00F031 EAEAEA          LDA    A,#data
TEST.ASM 127 ( TOTAL LINE 127 ) Error 20: Reference to undefined label or symbol 'data'
551 00F42B EAEA          BRA    TEST2
TEST.ASM 551 ( TOTAL LINE 551 ) Error 20: Reference to undefined label or symbol 'TEST2'
593 00F4FC EAEAEAEAEAEA    LDA    A,(work,x ; data set
TEST.ASM 593 ( TOTAL LINE 593 ) Error 23: '()' format error ';'

```

**Figure 2.8 TAG File Example**



# CHAPTER 3

## Source Program Coding Method

### 3.1 Structure of Source Program

A source program written in assembly language is made up of lines. Each source program line must comply with the following rules:

1. Each line must be complete by itself, and an instruction cannot be coded on more than one line.
2. Each line may contain no more than 256 characters. The assembler program ignores coding beyond 256 characters.
3. Each line consists of the following fields:
  - Symbol/label field  
Label for referencing this line from other locations or symbol whose value is to be set by the .EQU pseudo instruction is coded in this field.
  - Op-code/pseudo instruction field  
7700 Family instruction mnemonic (hereafter referred to as op-code) or pseudo instruction is coded in this field.
  - Operand field  
Object of processing by op-code or pseudo instruction is coded in this field.
  - Comment field  
Specification in this field is not processed by the assembler, and the user can use this field for any purpose.

There are five types of lines.

1. Instruction line  
An instruction line specifies a 7700 Family instruction. The assembler converts the specifications on this line to machine language data.
2. Structured preprocessor instruction line  
A structured preprocessor instruction line specifies the structured preprocessor language that is processed by PRE77.
3. Pseudo instruction line  
A pseudo instruction line specifies the information necessary for assembly.

### 4. Macro instruction line

A macro instruction line specifies the macro definition. This line is processed by the assembler.

### 5. Comment line

A comment line is not processed by the assembler. Therefore, it can be used by the user for any purpose.

## 3.2 Line Formats

This section describes the format of each type of line. The following conventions are used for these descriptions:

1.  $\triangle$  and  $\blacktriangle$  specify space or tab code.  $\triangle$  is required, and  $\blacktriangle$  is optional.
2. Colon (:) may be omitted when specifying a label, but if omitted, a space or a tab code must be specified between label and pseudo instruction.

### 3.2.1 Instruction Line

Shown below is the format of an instruction line:

$\blacktriangle$  Label:  $\blacktriangle$  Op-code  $\triangle$  Operand  $\blacktriangle$  ; Comment <RET>

$\blacktriangle$ † Op-code  $\triangle$  Operand  $\blacktriangle$  ; Comment <RET>

† Because RASM77 identifies each instruction by its reserved word, a line can begin with an opcode if there is no label.

### 3.2.2 Structured Preprocessor Instruction Line

This line is not processed by the assembler. Refer to Part 2. Chapter 3 for details concerning this line.

### 3.2.3 Pseudo Instruction Line

Shown below is the format of pseudo instruction line:

$\blacktriangle$ Label:  $\blacktriangle$  Pseudo-op  $\triangle$  Operand  $\blacktriangle$  ; Comment <RET>

$\blacktriangle$  Symbol  $\triangle$  .EQU  $\triangle$  Operand  $\blacktriangle$ ; Comment <RET>

$\blacktriangle$ ‡ Pseudo-op  $\triangle$  Operand  $\blacktriangle$  ; Comment <RET>

Notes:

‡ Because RASM77 identifies each pseudo instruction by its reserved word, a line can begin with an op-code if there is no label.

Also note that labels cannot be coded for some pseudo instructions. Refer to Chapter 5 and Appendix B for details.

### 3.2.4 Macro Instruction Line

The format of a macro instruction line is shown below. Refer to Chapter 6 and Appendix E for details concerning this line.

▲ Macro name:▲ Macro Instruction △ Operand ▲ ; Comment <RET>

Note:

If there are more than one data in the operand, they must be separated by a comma (.). Space or tab can be coded on both sides of a comma.

### 3.2.5 Comment Line

Comment line must begin with a semicolon (;). Shown below is the format of a comment line:

▲ ; Comment <RET>

## 3.3 Field Coding Method

### 3.3.1 Symbol/Label Field

RASM77 manages symbols and labels separately<sup>1</sup>, but the same name coding format applies to both. The coding format is described below.

1. A symbol or label can be specified using alphanumeric characters, special characters, underline (\_) and question mark (?). The first character must be an alphabetic or special character.
2. Reserved words cannot be used as names. RASM77 processes register names, flag names, op-code names, pseudo instruction names and operand description instructions (including DP and DT) as reserved words.
3. Uppercase and lowercase are recognized. Therefore, "BIG" and "Big" are recognized as different names.
4. A label or symbol may be no more than 255 characters long.
5. The following labels beginning with '..' (two periods) must not be used because they are labels generated by macro instructions '..' and PRE77<sup>2</sup>. Labels beginning with one or three periods are also prohibited.

---

<sup>1</sup> Names defined by the .EQU pseudo instruction or an instruction with the command parameter -D are treated as symbols, and other names are treated as labels.

<sup>2</sup> Preprocessor that processes structured code lines.

When specifying a label, it must be followed immediately by a colon (:). However, for compatibility with previous version, the colon may be omitted if the command option “-U” is specified. If the colon is omitted, a space or a tab code is required between the a label and pseudo instruction. It is recommended that this colon be always specified to make it easier to differentiate labels from symbols and to make label search by the editor more efficient.

### 3.3.2 Op-code/Pseudo Instruction Field

A 7700 Family instruction mnemonic or a pseudo instruction is specified in the op-code/pseudo instruction field. The specification format is described below.

1. No distinction is made between uppercase and lowercase characters for op-codes and pseudo instructions. Thus, both “NOP” and “nop” mean the same.

This field is described in more detail in Chapters 4 and 5.

### 3.3.3 Operand Field

Information regarding the target of op-code or pseudo instruction is specified in the operand field. The specification format is described below.

1. If there are two or more operand data, they must be separated by a comma (,).
2. Space or tab code may be specified on either side of a comma.

This field is described in more detail in Section 3.4.

### 3.3.4 Comment Field

Any user information may be specified in the comment field. The specification format is described below.

1. A comment field must begin with a semicolon (;).
2. Any character may be used in the comment field.

## 3.4 Operand Field Coding Method

### 3.4.1 Data Format

Operand field may be specified with data in any of the following four data formats:

1. Numeric constant

- A numeric constant can be specified as a positive or negative value by using the ‘+’ or ‘-’ instruction as prefix. If neither ‘+’ nor ‘-’ is specified, the numeric constant is processed as a positive value.
- A binary, octal, decimal or hexadecimal number may be specified as a numeric constant.
- When specifying a binary numeric constant, the value must be followed by ‘B’ or ‘b’.

Example: `.BYTE 100110B`

- When specifying an octal numeric constant, the value must be followed by 'O' or 'o'.  
Example: `.BYTE 70o`
- When specifying a decimal numeric constant, only an integer value can be specified.  
Example: `.BYTE 100`
- When specifying a hexadecimal numeric constant, the value must be followed by 'H' or 'h'. If the hexadecimal value begins with an alphabetic character (A to F), it must be prefixed with 0.  
Example 1: `.BYTE 64H`  
Example 2: `.BYTE 0ABH`

### 2. Character string constant

- Any ASCII code character may be used in a character string constant.
- Character string constant must be enclosed between single quotes ( ' ') or double quotes ( " ").  
Example: `.BYTE 'A'` => Sets 41H.
- If '\ ' is used in a string literal, the character immediately following the '\ ' is processed as character string data. When using '\ ' in a character string, write it as '\\ '.

### 3. Label or symbol

- A label has a 24 bit data value, and a symbol has a 32 bit data value.

### 4. Expression

- Numeric expression can be specified as a combination of instructions, numeric constants, character string constants, labels or symbols.
- An expression is calculated from left to right. (No operators priorities are recognized.)

Example 1:  $2 * 3$  => Result is 6.

Example 2:  $2 + 6 / 2$  => Result is 4.

## 3.4.2 Operators

Table 3.1 lists the operators that may be used with RASM77.

**Table 3.1 List of Operators**

Operator <sup>1</sup>	Description
+	Addition
-	Subtraction
*	Multiplication
/	Divide
%	Remainder of division
<<	Left shift
>>	Right shift
&	Logical AND on bits
	Logical OR on bits
^	Logical exclusive-OR on bits
+	Unary operator specifying a positive number
-	Unary operator specifying a negative number
~	Unary operator specifying bit inversion
@ <sup>2</sup>	Operator that converts the immediately following symbol to character string
SIZEOF <sup>3,4</sup>	Unary operator to obtain section size
BANK <sup>4</sup>	Unary operator to extract high-order 8 bits of label or symbol
OFFSET <sup>4</sup>	Unary operator to extract low-order 16 bits of label or symbol

Notes:

1. Operation is executed from left to right. (No operator priorities are recognized.)  
 Example 1:  $2+6/2 \Rightarrow$  Result is 4.  
 Example 2:  $2*3 \Rightarrow$  Result is 6.
2. The symbols concatenated by the @ operator must be absolute values. If a forward-referenced symbol is specified, an error occurs.
3. SIZEOF value is determined at link time regardless of whether the referenced section is relocatable or absolute. Accordingly, the same limitations as for external labels apply to the location where the SIZEOF instruction may be specified. (It cannot be specified in the operand of pseudo instructions such as .ORG and .BLKB.)
4. A space must be specified between SIZEOF, BANK, or OFFSET instruction and label, symbol or section name.  
 Example: `.DT BANK DATA1`

# CHAPTER 4

## Instruction Coding Method

### 4.1 Addressing Mode

The basic mode in which instructions specify the data to be processed is called an addressing mode. 7700 Family supports 28 addressing modes, and the operand coding format is prescribed for each of these addressing modes.

The following summarizes the characteristics of the addressing modes as they relate to operand coding:

1. Accumulator addressing mode

This addressing mode is for processing the data in an accumulator. The Series 7700 Family CPU has A and B accumulators, and the name of the accumulator to be used must be specified at the beginning of the operand field. Note that, if accumulator B is specified, the bytes count of machine language data will increase by 1 byte.

Example:       LDA A, #IMMDATA

2. Immediate addressing mode

This addressing mode is for directly specifying the data to be processed in the operand field. The value in the operand must be prefixed by "#".

Example:       LDM #IMMDATA, MEMORY

3. Direct addressing mode

This addressing mode is for using the 16 bit value in the direct page register (DPR) as the base address and specifying an 8 bit offset value to the base address in the machine language data. Bank address is fixed at 0. The method of coding direct addressing in assembler instructions is described in detail in Section 4.3.

4. Absolute addressing mode

This addressing mode is for using the 8 bit value in the data bank register (DT) as the base address and specifying the lower-level 16 bit address in the machine language data. The method of coding absolute addressing in assembler instructions is described in detail in Section 4.3.

5. Absolute long addressing mode

In this addressing mode, a 24 bit address value is specified in the operand field. (The entire memory space of 7700 Family can be used.) This addressing mode is used when the label specified in the operand field cannot be processed in the direct or absolute addressing mode. The method of coding absolute long addressing in assembler instructions is described in detail in Section 4.3.

## CHAPTER 4. INSTRUCTION CODING METHOD

---

### 6. Indexed addressing mode

In this addressing mode, address of the data to be processed is modified by the content of register X or Y. Register name must be specified after a comma (,).

Example:      ADC A, DATA1, X

### 7. Direct indirect addressing mode

In this addressing mode, the data to be processed is specified indirectly by memory address. The memory location where the 2 byte address of the data to be processed is stored is specified in the operand field. The high-order 8 bits contain the data bank register value. The memory address is specified in the operand field by the same method as for direct addressing. The operand value must be enclosed in parentheses.

Example:      LDA A, (INDATA)

When storing a 3 byte address in the memory location, "L", must be added at the end of the op-code.

Example:      LDAL A, (INDATA)

### 8. Absolute indirect addressing mode

In this addressing mode, the data to be processed is specified indirectly by memory location. The memory location where the 2 byte address of the data to be processed is stored is specified in the operand field. The low-order 16 bits of memory address is specified in the operand field, and the program bank register value is specified in the high-order 8 bits. The operand value must be enclosed in parentheses.

Example:      JUMP (PROCESS1)

When storing a 3 byte address in the memory location, "L" must be added at the end of the op-code.

Example:      JMPL (PROCESS2)

### 9. Relative addressing mode

In this addressing mode, the branch destination is specified by a relative byte count from the current program counter value. The relative value itself cannot be specified in the source program. If a label or object location is specified in the operand field, the assembler calculates the relative value.

Example:      BBA LABEL

Refer to Appendix C for the coding format of each addressing mode.



## 4.2 Data Length Specification

7700 Family CPU can control data length and index register length with the CPU internal flag. Table 4.1 summarizes the association between the flags and the assembler.

**Table 4.1 Association of CPU Internal Flags and Assembler**

- Data length selection flag (m)

Flag status	Meaning	Reset state
m = 0	16 bit operation	Reset state after CPU reset. This is the default value when RASM77 is started.
m = 1	8 bit operation	

- Index register length selection flag (x)

Flag status	Meaning	Reset state
x = 0	16 bits long	Reset state after CPU reset. This is the default value when RASM77 is started.
x = 1	8 bits long	

Because the machine language code for each instruction is identical regardless of the flags, flags do not affect assembler execution except in the case of immediate addressing. In the case of immediate addressing, the bytes count of the immediate value data that must be specified in the operand field depends on the data length. Therefore, the assembler must generate a code that is appropriate for the flag. RASM77 allows specification of status by one of two methods:

1. Direct specification in instruction's op-code

Example 1: `LDA.B A, #50H` ; Specifies an 8 bit immediate value (50H).

Example 2: `LDA.W A, #50H` ; Specifies a 16 bit immediate value (0050H).

2. Declaration of default value by pseudo instruction INDEX or .DATA

Example 1: `.INDEX 16` ; Declares that the index length is 16 bits.

Example 2: `LDX #200H` ; Specifies processing at the default index length (16 bits).

RASM77 allows for assemble control by default values of data and index lengths.

If a command parameter "-F" is specified, symbols "M\_FLAG" and "X\_FLAG" are handled as reserved words. If this command parameter is not specified, "M\_FLAG" and "X\_FLAG" can be used as any symbols or labels.

These reserved words can be used to verify the content of the flag 'm' or 'x' that is currently recognized by the assembler by using them along with a pseudo-instruction ".IF" that performs conditional assembling.

The functions of "M\_FLAG" and "X\_FLAG" described in Table 4.2.

**Table 4.2 Function of M\_FLAG and X\_FLAG**

Symbol	Function
M_FLAG	The value changes with the setup value of ".DATA." The value of "M_FLAG" = 1 when the data length is set to 16 bits. The value of "M_FLAG" = 0 when the data length is set to 8 bits.
X_FLAG	The value changes with the setup value of ".INDEX." The value of "X_FLAG" = 1 when the index length is set to 16 bits. The value of "X_FLAG" = 0 when the index length is set to 8 bits.

Figure 4.1 shows an example of a conditional assemble program using "M\_FLAG" and "X\_FLAG."

```
BIT8: .MACRO
      .DATA      8
      .INDEX     8
      .IF       M_FLAG
      sep       m, x
      .ELSE
      clp      m, x
      .ENDIF
```

**Figure 4.1 Example of conditional assemble based on data and index lengths**

Notes:

1. Pseudo instructions do not generate instructions that manipulate the CPU internal flags. Therefore, the user program must control the assembler's default value to be consistent with the processor status.
2. Immediate value must be used when specifying the data length directly in the operand of an instruction.

## 4.3 Setting Direct Page and Absolute Addressing

the 7700 Family CPU has two internal registers named direct page register (DPR) and data bank register (DT) to enable memory accessing with the least number of codes for each type of address space. By using these registers, 7700 Family's memory space (16M bytes) can be accessed more efficiently. The functions of DPR and DT are described below.

### 1. Direct page register (DPR)

DPR is a 16 bit register. At the machine language data level, direct addressing specifies the target address with an 8-bit offset to DPR. (Bank is always 0.)

If only a label is specified in the source program's operand field, the assembler checks whether the label is within offset values 00 to 0FFH from the current DPR value. When direct addressing is possible, the assembler calculates the offset value and generates machine language data.

### 2. Data bank register (DT)

DT is an 8 bit register. At the machine language data level, absolute addressing specifies a low-order 16 bit address with the value in DT as the bank address. If only a label is specified in the source program's operand field, the assembler checks if the current DT value and the label's high-order 8 bit value are identical. When absolute addressing is possible, the assembler calculates the low-order 16 bit value and generates machine language data. Table 4.3 summarizes the relations of DPR and DT to assembler.

**Table 4.3 Relationship between DPR, DT and Assembler**

Register	Reset state
DPR	0000H is set in DPR after CPU reset. This is the default value when RASM77 is started.
DT	00H is set in DT after CPU reset. This is the default value when RASM77 is started.

## 4.4 Addressing Mode Selection

RASM77 provides the following three addressing modes when a symbol, label, or absolute value is coded in the operand of an instruction:

1. Direct addressing mode
2. Absolute addressing mode
3. Absolute long addressing mode

## CHAPTER 4. INSTRUCTION CODING METHOD

---

RASM77 allows the addressing mode to be selected from these three modes. The method of selection depends on the symbol, absolute value, or label that is the target of the operation. For label operation, the value in the direct page register (DPR) and data bank register (DT) directly affects the selection of the addressing mode.

Described below are descriptions on how to set the DPR and DT registers followed by the description of addressing modes during symbol and absolute value, and label operations.

### 4.4.1 Setting the Direct Page Register and Data Bank Register

In order to change the direct page and bank to be used in RASM77, the value of the direct page register and data bank register must be declared with the .DP and .DT pseudo instructions beforehand as shown below.

Example:

```
.DP          100      ← Set 100 in direct page register (100H to 1FFH)
.DT          1        ← Set 1 in data bank register (bank 1)
LDA         A, #100H
TAD
LDT         #1
```

The operand of the .DP and .DT pseudo instructions can be either a numeric value specifying the direct page start address and bank address or a direct page name label and bank name label.

When referencing the labels DLAB and work coded in sample1.a77 from the PRO section of samp2.a77 as shown in the example below, the value of DPR and DT registers can be declared with the direct page name label DPR100 and bank name label BANK2.

Example:

[Source file for samp1.a77]

```
        .SECTION  DATA1
        .ORG      100H
DPR100:
DLAB:   .BLKW    2

        .SECTION  DATA2
        .ORG      20000H
BANK2:
work:   .BLKB    2
```

[Source file for samp2.a77]

```
.SECTION    PRO
.DPEXT     DPR100:DLAB
.DTEXT     BANK2:work

.DP        OFFSET DPR100 ← Set 100 in direct page register (100H to 1FFH)
.DT        BANK    BANK2 ← Set 1 in data bank register (bank 1)
LDA        A, #OFFSET DPR100
TAD
LDT        #BANK BANK2
```

The scope of the direct page and bank coded in the operand of .DP and .DT is determined during link.

### 4.4.2 Addressing Mode During Symbol•Absolute Value Operation

The addressing mode can be selected explicitly regardless of the value in the DPR and DT registers when public specification pseudo instruction .PUB and external reference specification pseudo instruction .EXT are used for data and one of the following addressing mode specifiers is used in the symbol•absolute value in the operand (except when OFF is specified as the operand of the .DP or .DT pseudo instruction.)

- DP: ..... Direct addressing
- DT: ..... Absolute addressing
- LG: ..... Absolute long addressing

Example:

```
.EXT        SYM1          ← Specified as public (.PUB) in another file

.SECTION    PRO
AND        A, DP:SYM1    ← Direct addressing
AND        A, DT:SYM1    ← Absolute addressing
AND        A, LG:SYM1    ← Absolute long addressing
```

If an instruction can select direct mode, absolute mode, or absolute long mode and uses a symbol that has an absolute value, the value is compared with the value in DPR and DT registers at the location of the instruction and the addressing mode that provides the most efficient memory usage is selected.

Example:

```
.SECTION      DATA
.ORG          100H
LAB1 .BLKW    1

.SECTION      PRO
.DT           0
.DP           100H
LDA          A, #100H
TDA
LDT          #0

AND          A, LAB1      ← Direct addressing
```

### 4.4.3 Addressing Mode During Label Operation

The addressing mode can be selected explicitly regardless of the value in the DPR and DT registers when public specification pseudo instruction .PUB and external reference specification pseudo instruction .EXT are used for data and one of the following addressing mode specifiers is used in the label in the operand (except when OFF is specified as the operand of the .DP or .DT pseudo instruction.)

- DP: ..... Direct addressing
- DT: ..... Absolute addressing
- LG: ..... Absolute long addressing

Example:

```
.EXT          LAB1      ← Specified as public (.PUB) in another file

.SECTION      PRO
AND          A, DP:LAB1 ← Direct addressing
AND          A, DT:LAB1 ← Absolute addressing
AND          A, LG:LAB1 ← Absolute long addressing
```

In addition, the following external reference specification pseudo instructions can be used to specify the addressing mode during data reference. In this case, the addressing mode specifier can be omitted in the operand.

- .DPEXT ..... Direct addressing
- .DTEXT ..... Absolute addressing
- .EXT ..... Absolute long addressing

Example:

.DPEXT	LAB1	← Specified as public (.PUB) in another file
.DTEXT	LAB2	← Specified as public (.PUB) in another file
.EXT	LAB3	← Specified as public (.PUB) in another file
.SECTION	PRO	
AND	A, LAB1	← Direct addressing
AND	A, LAB2	← Absolute addressing
AND	A, LAB3	← Absolute long addressing

If an instruction can select direct mode, absolute mode, or absolute long mode and the data has a relocatable value coded with the direct page name label or bank name label associated with .DPEXT or .DTEXT, the appropriate addressing mode is selected by comparing them with the direct page name label or bank name label coded with .DP or .DT.

Example:

.DPEXT	directpage_name:DPLAB	← directpage_name: Direct page name label
.DTEXT	databank_name:DTLAB	← databank_name: Bank name label
.SECTION	PRO	
.DP	OFFSET directpage_name	
.DT	BANK databank_name	
AND	A, DPLAB	← Direct addressing
AND	A, DTLAB	← Absolute addressing

### 4.4.4 Disabling Addressing Mode Selection

The addressing mode selected by RASM77 can be explicitly disabled. To disable the use of direct addressing mode, code OFF as the operand of .DT. To disable the use of absolute addressing mode, code OFF as the operand of .DP.

When direct addressing mode is disabled, absolute or absolute long addressing mode is selected. When absolute addressing mode is disabled, direct or absolute long addressing mode is selected.

If OFF is written in the operand of .DP or .DT, the rules described in Table 4.4 are followed when assembling the source.

**Table 4.4 Rules for addressing mode selection**

Description format	Processing by RASM77
DP:label	Error
DP:symbol	Selects direct addressing mode
DP:absolute value	Selects direct addressing mode
DT:label	Error
DT:symbol	Selects absolute addressing mode
DT:absolute value	Selects absolute addressing mode

Example:

```

        .DPEXT          DPLAB
        .SECTION        DATA
LAB:    .BLKB          1

        .SECTION        PRO
        .DP             OFF          ← Disable direct page addressing

        LDA             A, DPLAB     ← Absolute long addressing
        STA             A, DP:ADDR1  ← Error 22: Value is out of range
    
```



Note:

1. When using a relocatable local label in direct or absolute addressing, write "DP:" or "DT:" in the operand. If only a local label is written, the assembler uses absolute long addressing.
2. If a label specified with the pseudo instruction `.DPEXT` is coded in the operand and operation is performed on that label, direct addressing is also used for the code generated as the result of the operation.

Example:

```
.DPEXT  WORKA      ; ← Specify external label of a direct page
STA     A,WORKA+1 ; ← Treat operation result as direct page
```

In this case, the instruction `STA` is assembled using direct addressing. However, whether the operation result is within the scope of direct addressing is determined during linkage. The same is true for the label specified with `.DTEXT`.

3. If the command option “-Q” is specified, a warning is issued for instruction lines that specify other addressing mode with codes such as “LG:” for labels specified with `.DPEXT` or `.DTEXT`.

# CHAPTER 5

## Pseudo Instruction Coding Method

### 5.1 Function of Pseudo Instructions

A pseudo instruction declares or specifies<sup>1</sup> the assembler to generate the intended machine language data. RASM77 offers 52 pseudo instructions, and they can be grouped into the following five functional groups:

1. Assembly control pseudo instructions

- Does not generate data but controls generation of machine language data that corresponds to the instruction.
- Does not affect address updating.
- There are 12 assembly control pseudo instructions:

.INDEX, .DATA	Declares data length
.DT, .DP	Declares DT or DPR value
.IF, (ELSE), .ENDIF	Conditional assembly
.INCLUDE	Declares file to be included in program
.EQU	Equation
.END	Declares end of program
.ASSERT	Outputs a message
.ERROR	Declares assembly error
.DEFINE	Defines a string

---

<sup>1</sup> The term “declare” is used when a pseudo instruction specifies a default to the assembler and the term “specify” is used when a pseudo instruction specifies an instruction that affects the output file.

### 2. Location control pseudo instructions

- Updates address.
- Data definition pseudo instruction generates constant data.
- There are 10 pseudo instructions in this group:

.ORG	Declares location
.BLKB, .BLKW, .BLKA, .BLKD	Allocates RAM area
.BYTE, .WORD, .ADDR, .DWORD	Defines data
.EVEN	Corrects location alignment

### 3. Linkage control pseudo instructions

- Performs controls related to linkage processing.
- There are 8 pseudo instructions in this group:

.SECTION	Specifies section name
.DPEXT, .DTEXT, .EXT, PUB	Specifies global label name
.OBJ, .LIB	Specifies linkage filename
.VER	Specifies version

### 4. Listing control pseudo instructions

- Performs controls related to output of PRN file.
- There are 7 pseudo instructions in this group:

.PAGE	Specifies new page and title
.COL, .LINE	Specifies listing format (columns and row count)
.LIST, .NLIST	Outputs/suppresses list
.LISTM, .NLISTM	Outputs/suppresses macro expansion list

### 5. Source level debug support

- Outputs information necessary for source line debug to object file.
- There are 6 pseudo instructions in this group:

.CLINE	Outputs column information
.FUNC, .ENDFUNC	Specifies start/end of function
.LANGUAGE	Outputs the language used
.POINTER	Sets the pointer length
.SOURCE	Sets the source file name

### 6. Reserved pseudo instructions

- A number of pseudo instructions are reserved for future use. These instructions do not affect assembly.
- There are 9 pseudo instructions in this group:

.PROGNAME	Declares program name
.IO, .ENDIO, .RAM, .ENDRAM	Declares RAM area name
.PROCMAIN, .PROCSUB, .PROCINT, .ENDPROC	Declares module name

Functions of each pseudo instruction group are described in the next section.

# 5.2 Assembly Control Pseudo Instructions

## 5.2.1 Data Length Declaration

### .INDEX

Declares the default value for the index register length selection flag (x). For details, see Section 4.2.

### .DATA

Declares the default value for the data length selection flag (m). For details, see Section 4.2.

## 5.2.2 DPR and DT Value Declaration

### .DP

Declares the default value for the direct page register. The assembler selects the optimal addressing mode based on the DPR value specified for this and subsequent lines. For details, see Section 4.3.

### .DT

Declares the default value for the data bank register. The assembler selects the optimal addressing mode based on the DT value declared for this and subsequent lines. For details, see Section 4.3.

## 5.2.3 Conditional Assembly

### .IF, (ELSE), .ENDIF

Instructs the assembler to select the assembly location based on the symbol value. This pseudo instruction can be used to manage programs for different specifications by a single source program, to control assembly of test routines, etc.

## 5.2.4 Include File

### .INCLUDE

Instructs the assembler to include the contents of a file where this pseudo instruction is specified. This pseudo instruction is useful when editing a large source program in parts.

## 5.2.5 Equation

### .EQU

Defines an absolute value for a symbol. The same symbol can be redefined within the same program. If a forward reference is made to a symbol that is redefined, the last definition takes effect.

## 5.2.6 Declare End of Assembly

### .END

Declares the end of source program to be assembled. The assembler does not process any source data after this declaration.

### 5.2.7 Message Output

`.ASSERT`

Displays the specifies string on the screen.

### 5.2.8 Assembly Error Output

`.ERROR`

Displays the specifies string on the screen and stops assembly. The assembler does not process subsequent lines.

### 5.2.9 Define String

`.DEFINE`

Defines a string to a symbol.

## 5.3 Address Control Pseudo Instructions

### 5.3.1 Address Declaration

`.ORG`

Declares the address for the next line. The section in which this pseudo instruction is specified will have the absolute attribute, and address specification cannot be made for linkage processing. This pseudo instruction can be used in areas where the address is fixed such as an interrupt vector.

### 5.3.2 Memory Allocation

`.BLKB`, `.BLKW`, `.BLKA`, `.BLKD`

Allocates a RAM memory area of the size specified by operand.

### 5.3.3 Data Definition

`.BYTE`, `.WORD`, `ADDR`, `DWORD`

Generates data specified by operand in the ROM area.

### 5.3.4 Correct Address Alignment

`.EVEN`

Changes an odd numbered address to an even numbered address. Nothing occurs if the current address is even numbered. This pseudo instruction can be used to set the beginning of each data in a character string data in a byte-by-byte data area to an even numbered address.

# 5.4 Linkage Control Pseudo Instructions

## 5.4.1 Section Name Specification

`.SECTION`

Specifies the section name for the program that follows this line. RASM77 requires specification of a section name using this pseudo instruction at the beginning of every program.

## 5.4.2 Global Label Name Specification

`.DPEXT`, `.DTEXT`, `.EXT`

Specifies an externally referenced label or symbol name. The label names specified by these pseudo instructions must be specified as public labels in other files.

`.PUB`

Specifies that a label or symbol defined in this file can be referenced by other files.

## 5.4.3 Linkage Filename Specification

`.OBJ`, `.LIB`

Specifies the name of relocatable file to be linked or the name of library file. The files declared by these pseudo instructions are automatically referenced by the linker so that linkage command specification is simplified.

## 5.4.4 Version Control

`.VER`

Specifies the version name of a relocatable file. Version name consistency between relocatable files can be checked during linkage processing by specifying the version confirmation command parameter “-V”.

# 5.5 Listing Control Pseudo Instructions

`.PAGE`

Specifies new page and title of a listing.

`.COL`, `LINE`

Specifies the number of columns or rows for a listing. These pseudo instructions may be specified only one time each in a source file.

`.LIST`, `.NLIST`

Specifies control of listing output to a PRN file. These pseudo instructions should be used when only a portion of a listing is necessary as when partially debugging a program.

`.LISTM`, `.NLISTM`

Specifies whether to output macro expansion lines to PRN file.

---

## 5.6 Source Level Debug Support

### .CLINE

Sets the line number necessary for source debug.

### .FUNC, .ENDFUNC

Specifies the start and end of function (subroutine).

### .LANGUAGE

Sets the information concerning the language used.

### .POINTER

Sets the byte length of pointer variable used by C compiler.

### .SOURCE

Sets the source file name required for source debug.

#### Note:

Listed above are the pseudo-instructions output by PRE77 and C compiler. If these pseudo-instructions are written in the source program, the assembler may output an error.

## 5.7 Reserved Pseudo Instructions

Reserved pseudo instructions are reserved for future expansion of the assembler. If these instructions are specified in a source file, they will not cause errors. They also will not affect the assembly results. Reserved pseudo instructions may be used when checking a source file's contents by combining RASM77 and a commercially available character string search program.

Example: By specifying the pseudo instruction in .PROCMAIN a source file as shown below, a listing of main program names for all source files in the current directory can be generated by searching ".PROCMAIN" with a character string search program.

```
.PROCMAIN START_KEY_SCAN ; Key scan program entry.
```

# CHAPTER 6

## Macro Instruction

### 6.1 Macro Instruction Functions

Macro instruction enables programs written in series 7700 Family assembly language to be defined as a macro and used in a user source program by coding its name as an instruction in the operand field. This enables the 7700 Family to be used as an enhanced CPU during programming by creating various macro definitions in advance. In this way, the macro function enables the user to organize his own programming environment.

### 6.2 Macro Instruction Types

Macros can be classified into RASM77 provided macros and user defined macros.

#### 1. System macros

- REPEATI - .ENDM  
Repeats processing for the number of arguments specified in the operand.
- REPEATC - .ENDM  
Repeats processing for the number of characters specified as argument in the operand.
- REPEAT - .ENDM  
Repeats processing for the number of times specified in the operand.

#### 2. User macros

- .MACRO - .ENDM  
Defines a macro instruction.
- .EXITM  
Forces termination of macro expansion.
- .LOCAL  
Defines a label used within a macro as a local label.



### Notes:

1. System macros can be used by itself or within a user macro definition.
2. User macros must be defined before it can be used. Therefore, macro definitions are normally placed at the beginning of a program or included at the beginning with the `.INCLUDE` pseudo instruction. User macros can be nested up to 20 levels.
3. By providing macro definitions as separate files (macro library), they can be used simply by including them at the beginning of a program, thus eliminating the need to define them in each program.
4. Macro expansion lines are indicated with a '+' sign next to them in the source file list.
5. Labels declared as local are assembled with labels `..n` (n: 0 to 65535 in decimal) assigned in the order of appearance. Labels beginning with `..` are reserved for future use in RASM77. Therefore, labels beginning with `..` must not be used by the user.
6. Upper and lowercase characters are distinguished in macro names. Therefore, `MAC` and `Mac` are assumed to be two separate macros.

## 6.3 Macro Operators

Table 6.1 lists the operators that can be used in macro instructions.

**Table 6.1 List of Macro Operators**

Operators	Description
\ <sup>1</sup>	Placed before a special character (such as “”, “”, ‘\’) that cannot be used as macro argument to declare that character as argument. [Format] \character
;; <sup>2</sup>	Defines a comment within a macro definition that is not to be expanded. [Format] ;; comment
’ <sup>3</sup>	Used to enclose an argument in a macro call when the argument contains space, tab, comma, or a reserved word. [Format] “ character string”
\$ <sup>4</sup>	Used to concatenate a macro argument with a character string. [Format] (1) character string\$argument (2) argument\$character string

Notes:

1. Serves as an escape character and nullifies the special meaning of the character that follows.

Example:

[Macro Definition]

```
DATA:  .MACRO VAL
        .BYTE  VAL
        .ENDM
```

[Call example]

```
DATA  "\ "HELLO !\ "
```

[Macro expansion]

```
.BYTE  "HELLO !"
.ENDM
```

2. When the macro expansion result is output to a print file, comments beginning with ';' (one semicolon) is output in the macro expansion, but comments beginning with ';' (two semicolons) are not.

[Macro Definition]

```
LOOP:  .MACRO
        .LOCAL LOOP1
        LDA   A,#20      ; COMMENT
LOOP1: DEC   A           ;; COMMENT
        BNE   LOOP1     ;; COMMENT
        .ENDM
```

[Call example]

```
LOOP
```

[Macro expansion]

```
        LDA   A,#20      ; COMMENT
..0:    DEC   A
        BNE   ..0
        .ENDM
```

## CHAPTER 6. MACRO INSTRUCTION

---

3. If an argument in the operand contains space, tab, comma, or reserved word, the argument must be enclosed in "" (double quotes).

Example:

[Macro Definition]

```
SUB:  .REPEATI      INST, "NOP" , "LDA A, #1" , "RTS"
      INST
      .ENDM
```

[Macro expansion]

```
SUB:
      NOP
      LDA A, #1
      RTS
      .ENDM
```

4. The following code is allowed.

Example:

[Macro Definition]

```
W_LOAD:  .MACRO  MEM
         LDA    A, MEM$_L
         LDA    B, MEM$_H
         .ENDM
```

[Call example]

```
W_LOAD DATA
```

[Macro expansion]

```
LDA    A, DATA_L
LDA    B, DATA_H
.ENDM
```

# CHAPTER 7

## Operation

### 7.1 Starting RASM77

Before RASM77 can be executed, the following information (input parameters) must be input:

1. Source filename (required)
2. Command parameters

With RASM77, input parameters are input from a command line. Input parameters are described in Section 7.2, and the command line input method is explained by referring to examples in Section 7.3.

### 7.2 Input Parameters

#### 7.2.1 Source Filename

1. Name of source file to be assembled is specified. Source filename must always be specified. Only one source filename may be specified.
2. If specification of file extension (.A77) is omitted, .A77 is selected as default.
3. By specifying full filenames, files with other file extensions (e.g., .ASM) can be assembled by RASM77.
4. Filename can be specified with directory path. If only filename is specified, RASM77 processes a file in the current drive's current directory. The following example shows an example of assembling TEST.A77 in directory WORK on drive C.

Example: A>RASM77 C:\WORK\TEST<RET>

#### 7.2.2 Command Parameters

1. Command parameter may be specified in either uppercase or lowercase.
2. Each command parameter may be specified more than once at the same time. Each parameter must be delimited by a space.

Table 7.1 summarizes the functions of command parameters.

Table 7.1 List of Command Parameters

Command parameter	Description
-.	Suppresses output of all messages to the screen. This command parameter should be specified if no message display on the screen is desired as when executing an RASM77 from a batch execution file.
-A	Output no error message when any illegal descriptions are fined at condition is falt of conditional assemble instruction.
-B	Verifies the bit length <sup>1</sup> . When this parameter is specified, the bit length of local labels declared with pseudo instructions “.BYTE”, “.WORD”, “.BLKB”, or “.BLKW” are checked during reference with the bit length declared with “.DATA”, and “.INDEX” and warning 6 is issued if they do not match.
-C	Outputs source debug information to object file. Specify this option during assembly to perform source debug during debugging.
-D	Sets a numeric value for a symbol. The function is same as that of the pseudo instruction . EQU. Specification format is as follows (multiple symbols can be defined at one time by delimiting them by colon): -D symbol=numeric-value [ : symbol=numeric-value ... : symbol numeric-value] Example: A>RASM77 SRCFILE -DS1=10:S2=20<RET>
-E	Creates a TAG file and starts an editor. Editor’s program name is specified as follows: -E[editor-name] Example: A>RASM77 SRCFILE -EMI<RET> The item in [ ] may be omitted, and, if omitted, only TAG file creation is performed. When an editor name is specified, the editor is started after termination of assembly by using the TAG file that has been created as the argument. If no error occurs, editor is not started.
-F	The symbol "M_FLAG" or "X_FLAG" that holds the flag status of 'm' or 'x' that is currently assumed by RASM77 is made usable. By using this command parameter to perform conditional judgment of "M_FLAG" or "X_FLAG" with '8' or '16' in the operand of conditional assemble pseudo-instruction ".IF," the status of the above flag assumed by RASM77 is determined (whether 8 bits or 16 bits long), making it possible to separate processing in a macro, etc.
-L	Creates a PRN file. PRN file is not created unless this command parameter is specified.
-LC	Outputs to PRN file the false condition part when performing conditional assembly with .IF instruction, and the conversion string in @ instruction and .DEFINE pseudo instruction. These items are not output to the PRN file if this parameter is not specified.
-LD and -LD0	For the lines using the ".DEFINE" pseudo-instruction and "@" operator, only the replaced result lines are output to a print file.
-LD1	For the lines using the ".DEFINE" pseudo-instruction and "@" operator, both the lines before replacement and the replaced result lines are output to a print file.
-M	Outputs macro expansion to PRN file and generates the PRN file. Macro expansions are not output to PRN file if this parameter is not specified.

## 7.2 Input Parameters

Command parameter	Description
-N	Suppresses output of symbol list at the end of PRN file.
-O	Specifies the output destination path for the file to be created. Either directory or drive name may be specified as the path. If this command parameter is not specified, the file created is output to the same path as that of the source file. Specification format is as follows: -Opath-name Example: A>RASM77 SRCFILE -OB:\WORK<RET>
-P	Line numbers are not increase in macro expansion.
-Q	Outputs warning 7 when the pseudo instruction ".EQU" is used to equate a symbol that is already equated. No warning is issued when the same symbol is equated to a different value if this parameter is not specified. Warning 8 is issued if "LG:" is coded to selected absolute long addressing mode. No warning is issued when "LG:" is coded if this parameter is not specified.
-S	Outputs local symbol information to object file. Specify this option during assembly to debug local symbols during debugging <sup>3</sup> .
-T	If an error occurs in the user macro, the line number information of the macro call line, and not a macro definition line, is output to a tag file.
-U	Allows colon (;) following a label to be omitted.
-V	The version No. of RASM77 is output to the screen and the command is terminated.
-X	Starts the cross-referencer CRF77 when assembly terminates <sup>4</sup> . Example: A>RASM77 SRCFILE -X<RET>

### Notes:

1. Bit length cannot be changed with instructions such as "SEM" or "CLM". This is not appropriate for external reference. The x flag is checked for addressing mode other than immediate addressing mode of "CPX", "CPY", "LDX", "LDY", "STX", and "STY". m flag is checked for direct, direct indexed X, indexed Y, absolute, absolute indexed X, absolute indexed Y, absolute long, absolute long indexed X, direct bit, and absolute bit addressing mode.
2. An editor is started indirectly via the MS-DOS COMMAND.COM file, so that existence of COMMAND.COM file in the MS-DOS command path must be verified in advance. When working on a drive other than that in which COMMAND.COM file resides, the following specification must be made in the CONFIG.SYS file:

Example) SHELL=A:\COMMAND.COM A:\ /P

When the editor cannot be found in the current directory or command path, MS DOS will output an error message on the screen.

3. RASM77 does not output local symbol information if this option is not specified. Use this option if local symbol information is necessary during debugging.
4. A system error will occur if CRF77 is not in the current directory or command path.

### 7.3 Input Method

RASM77 is started by entering a command line after the prompt. Figure 7.1 illustrates entry of RASM77 startup command.

```
RASM77 TESTNAME -L -E <RET>
      ↑         ↑
Name of source  Command parameters
to be assembled
```

**Figure 7.1 Example of RASM77 Startup Command Line**

If input error is detected on command line input, a HELP screen is displayed as shown in Figure 7.2 and assembly is canceled.

When a command line is input correctly, assembly begins. When assembly is completed, number of errors, number of warnings, total number of lines assembled, number of comment lines and memory size for each section are displayed on the screen. Figure 7.3 shows an example of screen display when assembly has terminated normally.



```
A>RASM77<RET>
7700 Family RELOCATABLE ASSEMBLER V.5.00.00
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Usage: rasm77 <filename> [options]
  -. : all messages suppressed
  -a : don't care error at conditional assembly FALSE block
  -b : bit length check
  -c : source line information output to .r77 file
  -d : define symbol ( syntax -dSYMBOL1=DATA1:SYMBOL2=DATA2 )
  -e : make tag file
  -f : x, m flag information symbol enable
  -l : make list file
  -lc: conditional assembly statements output to .prn file
  -ld: .DEFINE statements output control
  -m : macro expansion statements output to .prn file
  -n : don't make symbol list to .prn file
  -o : select drive and directory for output ( syntax -o/tmp )
  -p : print file number control
  -q : .EQU symbol multi define warning message output
  -s : local symbol data output to .r77 file
  -t : tag-file form change
  -u : don't care ':' at end of label
  -v : rasm77 version display
  -x : execute crf77
```

Figure 7.2 HELP Screen for Command Line Error

```
>RASM77<RET>
7700 Family RELOCATABLE ASSEMBLER V.5.00.00
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.
now processing pass 1 ( TEST.A77 )
---*---
now processing pass 2 ( TEST.A77 )
---*---

ERROR   COUNT      00000   (0000H)
WARNING COUNT     00000   (0000H)
TOTAL   LINE       00994   (03E2H) LINES
COMMENT LINE      00247   (00F7H) LINES
WORK                00000010 (000000AH) BYTES
DATA                00000054 (000036H) BYTES
PROG                00001938 (000792H) BYTES

A>
```

**Figure 7.3 Normal Termination Screen**

---

## 7.4 Errors

### 7.4.1 Error Types

The following types of errors may occur during execution of RASM77:

1. OS errors

Errors related to the environment in which RASM77 is executed. These errors include disk and memory shortages. When such an error occurs, the error message list in Appendix A should be checked and the appropriate OS command should be entered.

2. RASM77 command line input errors

These are the errors in RASM77 startup command line input. Command line input should be checked against the descriptions in this chapter, and a correct command line must be re-input.

3. Assembly source file contents errors

These are errors in the contents of the source file being assembled such as duplicate label definition and referencing of undefined symbol. The source file errors must be corrected, and the source file must be reassembled. When an assembly error is detected, RASM77 does not create a relocatable file.

When RASM77 detects an error or warning condition, it outputs error information in the format shown in Figure 7.4 (filename, line number in file, sequential line number, error number and error message) on the screen and to PRN file. The information should be checked against the error message list (in error number order) in Appendix A, and appropriate action must be taken.

```
A>RASM77 TEST<RET>
7700 Family RELOCATABLE ASSEMBLER V.5.00.00
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.
now processing pass 1 ( TEST.A77 )
----*----
now processing pass 2 ( TEST.A77 )
-
  115 00F025 EAEA BCC      LOOP2
TEST.ASM 115 ( TOTAL LINE 115 ) Error 18: Relative jump is out of range
  127 00F031 EAEAEA      LDA      A,#data
TEST.ASM 127 ( TOTAL LINE 127 ) Error 20: Reference to undefined label or symbol "data"
---*
  551 00F42B EAEA BRA      TEST2
TEST.ASM 551 ( TOTAL LINE 551 ) Error 20: Reference to undefined label or symbol "TEST2"
  593 00F4FC EAEAEAEAEAEA LDA      A,(work,x ; data set
TEST.ASM 593 ( TOTAL LINE 593 ) Error 23: "(" format error ";"
----

ERROR   COUNT      00004   (0004H)
WARNING COUNT      00000   (0000H)
TOTAL   LINE       00994   (03E2H) LINES
COMMENT LINE       00247   (00F7H) LINES
WORK                00000010 (00000AH) BYTES
DATA                00000053 (000035H) BYTES
PROG                00001938 (000792H) BYTES

A>
```

**Figure 7.4 Error Display Example**

### 7.4.2 Return Values to OS

When using an OS batch execution file, there are times when it is desirable to modify processing according to results of execution. RASM77 returns one of four error level values to OS depending on the result of execution as summarized in Table 7.2. For explanation of how to utilize these error level return values, refer to an OS reference guide

Table 7.2 Listing of Error Levels

Error level	Execution result
0	Normal termination
1	Assembly source file contents error
2	RASM77 command input error
3	OS error
4	Force termination by ^C (control C) input

## 7.5 Environment Variables

RASM77 uses the following MS-DOS environment variables:

#### 1. TMP77

This variable specifies the name of the directory in which temporary files are created during assembly. If this environment variable is not set, the temporary files are created in the current directory.

Example:     SET TMP77=A:\TMP

#### 2. INC77

This variable specifies the directory of the files included during assembly. If the file specified with the .INCLUDE pseudo instruction cannot be found, it is loaded from this directory. However, this environment variable is ignored if the ".INCLUDE" operand specifies a path.

Example:     SET INC77=A:\INCLUDE

# APPENDIX A

## Error Messages

### A.1 System Error Messages

When a system error is detected during assembly, RASM77 outputs an error message on the screen and cancels assembly. Table A.1 lists the system error messages.

**Table A.1 List of System Error Messages**

Error Message	Description and User Action
Usage: rasm77 <filename> [-.] [-dSYMBOL1=DATA1] [-e] [-l] [-oPATH] [-x]	Command input is invalid.  ⇒ Check the HELP screen, and reenter the command.
Can't open xxx	File cannot be found.  ⇒ Check the source filename, and reenter correctly.
Can't create xxx	File cannot be created.  ⇒ Check the -o parameter specification, and reenter correctly.
Out of disk space	Disk space is insufficient for file output.  ⇒ Provide sufficient free space on the disk.
Out of heap space	Memory space is insufficient to execute the assembler.  ⇒ Reduce the number of symbols or labels.
Can't find crf77.exe	CRF77 cannot be found.  ⇒ Copy CRF77 to the current directory or a directory specified by MS-DOS command path.
Can't find command.com for execute xxx	COMMAND.COM file necessary to start the editor specified by option cannot be found.  ⇒ Check MS-DOS command path specification.

## A.2 Assembly Error Messages

When an assembly error is detected, RASM77 outputs an error message to the screen and to a PRN file. Table A.2 lists the assembly error messages.

**Table A.2 List of Assembly Errors**

Error No.	Error Message	Meaning and Actions
1	Already had same statement	A pseudo instruction that can be used only once in a source file is used two or more time. Example:       LINE 60 : LINE 80 ⇒ Correct declaration to only one.
2	Reference to forward label or symbol	A pseudo instruction is referencing a label or symbol that is defined later. Example:        .ORG TOP TOP: ⇒ Define the label or symbol before it is referenced.
3	Division by 0	Arithmetic expression includes division by 0. ⇒ Check the arithmetic expression.
4	Illegal operand	Operand specification contains illegal character. Example:        LDA A, # \$10 ⇒ Check operand specification.
5	Improper operand type	Mnemonic and operand combination is invalid. Example:        ADC.B A, work ⇒ Check instruction specification format.
6	Invalid label definition	Label is defined where it is not allowed. Example 1:     LABEL1: LINE 60 Delete the definition label. Example 2:     LABEL2: .EQU 100 ⇒ Change the label to a symbol.



## A.2 Assembly Error Messages

Error No.	Error Message	Meaning and Actions
7	Invalid symbol definition	Symbol is defined where it is not allowed. Example:       SYMBOL     LINE 60 ⇒ Delete the symbol definition.
8	Out of maximum program size	Address exceeds 0FFFFFFFH. Example:       ORG         0FFFFFF0H .WORD       1,2,3,4,5,6,7,8,9 ⇒ Modify the program so that address will be within the permitted range.
9	Label or symbol is multiple defined	Same label or symbol is defined more than once. Example:       MAIN:       NOP MAIN:       NOP ⇒ Check the label or symbol name.
10	Nesting error	The pseudo instruction INCLUDE is nested. ⇒ Modify the program so than the pseudo instruction is not nested.
11	No .END statement	.END statement is missing in source file. ⇒ Specify the END statement at the end of program.
12	No symbol definition	Symbol is not specified. Example:       .EQU 60 ⇒ Specify the symbol
13	No ';' at the top of comment	Comment field specification does not begin with a semi-colon (;) Example:       LDA A,#CNT counter set ⇒ Specify ';' at the beginning of comment field.

## APPENDIX A. ERROR MESSAGES

Error No.	Error Message	Meaning and Actions
14	Not in conditional block	ELSE or .ENDIF statement is specified without .IF statement. (This error also occurs when the associated .IF statement is erroneous.) Example:        IF DATA : .ENDIF : ELSE : .ENDIF ⇒ Check the .IF statement specification.
15	Operand is expected	Required operand is missing for an instruction. Example:        .BYTE ⇒ Check the operand specification.
16	Questionable syntax	A mnemonic is misspelled. Example:        ADD A, #DATA ⇒ Check the spelling.
17	Reference to multi defined label or symbol	Duplicate label or symbol is referenced. Example:        MAIN:        NOP MAIN:        NOP BRA   MAIN ⇒ Check the label or symbol name.
18	Relative jump is out of range	Relative jump instruction's destination address is out of range. ⇒ Rearrange the program, or change the jump instruction.
19	Label or symbol is reserved word	Register's name is used as a label or symbol. Example:        A .EQU   1FFH ⇒ Change the label or symbol name.
20	Reference to undefined label or symbol	Undefined label or symbol is referenced. ⇒ Check the label or symbol.

## A.2 Assembly Error Messages

Error No.	Error Message	Meaning and Actions
21	Value error	Data specification format is invalid. Example:     ADC A, #'A ⇒ Check the data specification format.
22	Value is out of range	Data is out of range. Example:     ADC.B     A, #100H ⇒ Check the operand specification format.
23	"()" format error	Numbers of left and right parentheses are not equal. Example:     ADC A, (WORK ⇒ Check the operand specification.
24	Relocatable error	Pseudo instruction .ORG is specified in the relocatable section. Example:     .SECTION PROG LDA A, WORK : .ORG 1000H LDA A, WORK ⇒ Make section division. A relocatable value is coded as the operand of a pseudo definition instruction .EQU. Example:     .EXT WORK0 .EXT WORK1 SYMBOL .EQU WORK0-WORK1 ⇒ Code a local value as the operand of pseudo instruction .EQU.
25	No SECTION statement	Pseudo instruction SECTION is not specified. ⇒ Specify SECTION statement in front of the program.
26	Reference to undefined section	Undefined section name is referenced. Example:     LDA A, #SIZEOF UNDEF_SECT ⇒ Check the indicated section.

## APPENDIX A. ERROR MESSAGES

Error No.	Error Message	Meaning and Actions
27	Page error	<p>Direct page name or data bank name specified by external referencing specifying pseudo instruction .DPEXT or .DTEXT is not equal to the current DPR or DT value.</p> <p>Example:</p> <pre>.DPEXT    PG1  :LABEL .DP       BANK PG2 LDA       A,  LABEL</pre> <p>⇒ Either specify the DPR or DT value for external referencing declaration or delete the direct page name or data bank name from the operand of .DPEXT or .DTEXT. (In the latter case, processing is executed using the current DPR or DT value specified by the pseudo instruction .DP or .DT.)</p>
28	Section type mismatch	<p>Instruction or data definition pseudo instruction (e.g., .BYTE) and memory allocation instruction (e.g., .BLKB) are specified in same section.</p> <p>Example:</p> <pre>LDA       A, #WORK .BLKB     1</pre> <p>⇒ Split the section.</p>
29	Function is multiple defined	<p>The name specified with .FUNC is defined more than once.</p> <p>Example:</p> <pre>.FUNC     FUNC_1 .FUNC     FUNC_1</pre> <p>⇒ Check the label name.</p>
30	Macro nesting error	<p>The macro instruction nesting level limit is exceeded.</p> <p>Example:</p> <pre>MAC : MACRO    DATA       LDA      A, DATA       MAC2</pre> <p>⇒ Reduce the macro instruction nesting level.</p>
31	No .ENDM statement	<p>There is no .ENDM statement in the source file.</p> <p>⇒ Code a .ENDM statement at the end of the macro definition.</p>
32	Illegal mnemonic	<p>The written mnemonic does not match the MCU.</p> <p>⇒ Specify the correct MCU type by using the .MCU pseudo-instruction.</p>
33	Illegal processor type	<p>The MCU type written in the operand of .MCU is incorrect.</p> <p>⇒ Specify the correct MCU type in the .MCU operand.</p>

## A.3 Warning Messages

When a warning condition is detected, RASM77 outputs a warning message to the screen and to a PRN file. Table A.3 lists the warning messages.

## APPENDIX A. ERROR MESSAGES

Table A.3 List of Warning Messages

Warning No.	Warning Message	Description and User Action
1	Phase warning	<p>1) Pseudo instruction .ORG specifies an address smaller than the previous address.</p> <p>Example:</p> <pre> .ORG      0E000H MAIN:    LDA      A, WORK       :       .ORG      0C000H </pre> <p>2) Instruction references a label or symbol that is defined later. (This warning message is output only for absolute section.)</p> <p>Example:</p> <pre>       LDA      A, TBL, X       : TBL:    .BYTE    0, 1, 2, 3, 4 </pre> <p>⇒ Either define the label or symbol before the line that references it, or specify DP: or DT: in operand.</p>
2	.END statement in include file	<p>Pseudo instruction END is, specified in an include file.</p> <p>⇒ Specify the END instruction in the source file.</p>
3	Line number is out of range	The operand of a .CLINE instruction exceeds 9999.
4	Too many actual macro parameters	The number of actual parameters for a macro call is greater than the number of dummy parameters.
5	Too few actual macro parameters	The number of dummy parameters for a macro call is less than the number of actual parameters.
6	Bit length is different from label or symbol	Length of label or symbol reference is different from its definition.
7	.EQU symbol is multiple defined	The same symbol is used more than once in a .EQU pseudo instruction.
8	Addressing mode warning	Addressing mode other than the one specified with the label is used.
9	Value warning	An invalid operand is specified as operand.

# APPENDIX B

## Pseudo Instructions

### B.1 Conventions

Pseudo instructions that can be used with RASM77 are explained alphabetically. The following conventions are used in describing each pseudo instruction:

1. Item in [ ] may be omitted.
2. Space or tab code is indicated by  $\triangle$  or  $\blacktriangle$ .  $\triangle$  is a required space or tab code, and  $\blacktriangle$  is an optional space or tab code (i.e., may be omitted).
3.  $\blacktriangle$  is used to separate a label from pseudo instruction. When specifying a label, a colon (:) is not required, but if it is omitted, either a space or a tab code must be specified between the label and pseudo instruction.

### B.2 Pseudo Instructions

### **.ADDR**

Define address data (3 byte)

---

**Format:**

▲[label:]▲.ADDR △expression

**Description:**

- Defines 3 byte constant data.
- Data is defined from low-order byte (it is defined from high-order byte in the OBJ field of the PRN file).
- Up to 8 data can be defined, but each data must be delimited by ','.
- A global label may be defined in the operand field.

**Example:**

```
TABLE: .ADDR    subl    ; Defines value of subl from low-order byte.
```

### **.ASSERT**

Output message

---

**Format:**

▲.ASSERT △'character-string'

**Description:**

- This instruction is used together with conditional assembly instruction (.IF).
- Outputs the specified character string to the screen.

**Example:**

```
.IF    JAPAN
.ASSERT 'Assembly with domestic specification'
:
.ELSE
.ASSERT 'Assembly with international specification'
:
.ENDIF
```



---

**.BLKA****Allocate RAM area (in units of 3 bytes)**

---

**Format:**

▲[label:]▲.BLKA△expression

**Description:**

- Allocates a RAM area of the specified size in units of 3 bytes.
- Label or symbol used in expression must be defined before this line.
- Label with relocatable value may not be specified in the operand field.

**Example:**

```
label:    .BLKA    10    ; Allocates a 30 byte RAM area.
```

---

**.BLKB****Allocate RAM area (in units of 1 byte)**

---

**Format:**

▲[label:]▲.BLKB△expression

**Description:**

- Allocates a RAM area of the specified size in units of 1 byte.
- Label or symbol used in expression must be defined before this line.
- Label with relocatable value may not be specified in the operand field.

**Example:**

```
label:    .BLKB10    ; Allocates a 10 byte RAM area.
```

### **.BLKD**

Allocate RAM area (in units of double-word)

---

**Format:**

▲[label:]▲.BLKD△expression

**Description:**

- Allocates a RAM area of the specified size in units of double word.
- Label or symbol used in expression must be defined before this line.
- Label with relocatable value may not be specified in the operand field.

**Example:**

```
label:      .BLKD10      ; Allocates a 40 byte RAM area.
```

### **.BLKW**

Allocate RAM area (in units of word)

---

**Format:**

▲[label:]▲.BLKW△expression

**Description:**

- Allocates a RAM area of the specified size in units of word.
- Label or symbol used in expression must be defined before this line.
- Label with relocatable value may not be specified in the operand field.

**Example:**

```
label:      .BLKW10      ; Allocates a 20 byte RAM area.
```

---

**.BYTE**Define 1-byte data

---

**Format:**

▲[label:]▲.BYTE△expression

**Description:**

- Defines 1-byte constant data.
- Up to 255 data may be specified, but each data must be delimited by ‘,’.
- Character strings must be enclosed in single quotes ‘’.
- When using ‘\’ in a character string, write it as ‘\\’.
- Global label may be specified in the operand field.

**Example:**

```
label1: .BYTE 10 ; Defines 0AH.
label2: .BYTE 'A','B' ; Defines 41H and 42H.
label3: .BYTE 'ABCD' ; Defines 41H, 42H, 43H and 44H.
```

---

**.COL**Specify columns count (default is 132)

---

**Format:**

▲.COL△expression

**Description:**

- Specifies the number of characters per listing line (80 to 132).
- 80 is assumed if 79 or smaller value is specified; 132 is assumed if 133 or larger value is specified.
- This pseudo instruction can be specified only once in a program.

**Example:**

```
.COL 100 ; Sets 100 column line.
```

**.DATA**

Declare data length (default is 16)

---

**Format:**

▲.DATA△expression

**Description:**

- Declares the internal CPU data length (8 or 16). Indicates an 8 bit data if the value of expression is 8; 16 bit data if value of expression is 16.
- This pseudo instruction affects the data length of addressing modes related to the M flag.
- New data length must be declared by this pseudo instruction when changing data length by the SEM or CLM instruction.
- Data length can be specified by adding the prefix “.B” or “.W” to op-code. If the data length specified by op-code differs from that specified by this pseudo instruction, program is assembled using the op-code specified data length.
- Note that this pseudo instruction only declares the data length for the assembler, and it does not manipulate the data length selection flag (m) for the internal CPU processor status register.

**Example:**

```
SEM                ; Sets M flag.  
.DATA8            ; Specifies data length.  
ADC  A,#DATA      ; Executes 8 bit addition.
```

---

**.DEFINE**Define a character string

---

**Format:**

symbol△.DEFINE△character-string

**Description:**

- Defines a character string to a symbol.
- The string must be enclosed in single ( ' ') or double ( " ") quotes if it contains space or tab characters.
- Character strings defined within structure preprocessor code cannot be used in the operand.
- The data after replacement is output to the print file. However, if the command option "-LC" is specified, the data before replacement is output.
- Symbol replacement is performed before macro expansion.

**Note:**

- The symbol defined with .DEFINE can only be used within the file where it is defined (it cannot be used as the operand of the .EXT, .DPEXT, .DTEXT or .PUB).

**Example:**

```
FLAG1 .DEFINE "#01H,DATA1" ; Set bit pattern 01H of DATA1 to FLAG1.  
;  
CLB FLAG1 ; Clear lowermost bit of DATA1.
```

### **.DP**

Declare direct page register value (default is 0000H)

---

**Format:**

▲.DP△numeric value or OFF

**Description:**

- Declares the direct page register (DPR) value (00000H-FFFFH).
- To change DPR value in a program, use this pseudo instruction to declare the new value.
- If “OFF” is specified in the operand field, the assembler does not use the direct addressing mode. (Selects either absolute or absolute long addressing mode.)

**Note:**

- This pseudo instruction only declares the value of the direct page register to the assembler. The actual value of the direct page register is unchanged.

**Example:**

```
.DP 1000H ; Declares 001000-0010FFH as DPR value.  
.DP OFF ; Declares not to use direct addressing mode
```

### **.DPEXT**

Declare external reference (direct page)

---

**Format:**

▲.DPEXT△[direct-page-name-label:]label[,label,...,label]

**Description:**

- Declares external referencing of the labels specified in the operand field in the direct page addressing mode.
- If a direct page name label is specified, the assembler uses the value of this label as the direct page value. The instruction “OFFSET” must not be specified.
- If direct page name is not specified, the assembler executes processing assuming that the label is included in the current DPR that has been declared by the pseudo instruction .DP.
- This pseudo instruction must be specified before the line referencing the label.

**Example:**

```
.DPEXT DRPG1 : WORK1 , WORK2 , WORK3
```

---

**.DT**Declare data bank register value (default is 00H)

---

**Format:**

▲.DT△numeric value or OFF

**Description:**

- Declares the data bank register (DT) value (00H-0FFH).
- Use the instruction BANK to change the bank using the label in the source program.
- To change DT value in a program, use this pseudo instruction to declare the new value.
- If "OFF" is specified in the operand field, the assembler does not use the absolute addressing mode. (Selects either direct or absolute long addressing mode.)

**Note:**

- This pseudo instruction only declares the value of the direct page register to the assembler. The actual value of the data bank register is unchanged.

**Example:**

```
.DT 01H           ; Declares 010000-01FFFFH as DT value.  
.DT BANK LABEL   ; Sets the high-order 8-bits of LABEL as the  
                 ; value of data bank register  
.DT OFF          ; Declares not to use absolute addressing mode.
```

### **.DTEXT**

Declare external reference (data bank)

---

**Format:**

▲.DTEXT△[bank-name-label:]label[,label,...,label]

**Description:**

- Declares external referencing of the labels specified in the operand field in the absolute addressing mode.
- If a bank name label is specified, the assembler uses the value of this label as the data bank value. The instruction "BANK" is not necessary.
- If bank name is not specified, the assembler executes processing assuming that the label is included in the current DT value that has been declared by the pseudo instruction .DT.
- This pseudo instruction must be specified before the label referencing line.

**Example:**

```
.DTEXT      DTPG1:WORK1,WORK2,WORK3
```

### **.DWORD**

Define double-word data

---

**Format:**

▲[label:]▲.DWORD△expression

**Description:**

- Defines 1-double-word data.
- Up to 8 data may be specified, but each data must be delimited by ','.
- Data is defined from low-order byte.
- Global label may be specified in the operand field.

**Example:**

```
label: .DWORD 0E1000H ; Defines 00H, 10H, 0EH and 00H.  
      .DWORD symbol  ; Defines values of 'symbol' from low-order byte.
```



---

**.END**Declare end of program

---

**Format:**

▲.END

**Description:**

- Declares the end of a source program.
- This pseudo instruction and subsequent lines are not assembled.

**Example:**

```
END ; Declares end of program.
```

---

**.EQU**Equation

---

**Format:**

symbol△.EQU△expression

**Description:**

- Equates a numeric value (double-word value) to a symbol.
- Label or symbol used in expression must be defined before this line.
- Label with relocatable value may not be specified in the operand field.
- If the command option “-Q” is specified, a warning is issued when an attempt is made to equate a symbol that has already been equated.

**Note:**

- If a numerical expression is written in a symbol where the result may take on a double-word or greater value, the value of 4 low-order bytes becomes effective and no error is output.

**Example:**

```
SYMBOL .EQU 1 ; Equates SYMBOL to 1.  
:  
SYMBOL .EQU 3 ; Equates SYMBOL to 3.  
:  
SYMBOL .EQU SYMBOL+7 ; Equates SYMBOL to 10.
```

### **.ERROR**

Declare an assembly error

---

**Format:**

▲.ERROR△'character-string'

**Description:**

- This instruction is used together with conditional assembly instruction (.IF).
- This instruction outputs the string specified as the operand to the screen and terminates assembly if an invalid condition is specified.

**Example:**

```
.IF      MODE
:
.ELSE
.ERROR  'Undefined assemble mode'
.ENDIF
```

### **.EVEN**

Correct address alignment

---

**Format:**

▲.EVEN

**Description:**

- Corrects address to an even-numbered address.
- EAH is output when this pseudo instruction is used in a ROM attribute section (section for instruction or data definition instruction); address update only is performed when this pseudo instruction is used in a RAM section (memory allocation instruction section). Nothing is performed if the address to be corrected is even.

**Example:**

```
.BYTE01H ; Defines 01H data.
.EVEN    ; Corrects data location alignment (outputs EAH code).
```

**.EXT**

Declare external reference

**Format:**

▲.EXT△label or symbol[label or symbol,...,label or symbol]

**Description:**

- Declares external referencing of the labels (or symbols) specified in the operand field.
- Direct or absolute addressing code is generated by “:DT” or “:DP” for external label referenced by “.EXT”. If the command option “-Q” is specified, a warning is issued where “:DT” and “:DP” are specified.
- An error will occur during linkage if the scope of direct or absolute addressing mode is exceeded.
- This pseudo instruction must be specified before the lines that reference the labels or symbols.

**Note:**

- If the label externally referenced by this pseudo instruction is used as operand other than memory reference (such as destination of JMP instruction or for immediate addressing mode), the low-order value of the label is used and no error occurs.

**Example:**

```
.EXT WORK1,WORK2,WORK3

.EXT E_LABEL          ; Declares external reference label
:
LDA  A,E_LABEL        ; Absolute long addressing
:
LDA  A,DP:E_LABEL     ; Direct addressing
:
LDA  A,DT:E_LABEL     ; Absolute addressing
```

**Format:**

```
▲.IF△expression  
<statement-1>  
▲.ELSE  
<statement-2>  
▲.ENDIF
```

**Description:**

- Assembles statement-1 if the expression that follows .IF is true (not 0 or character string data); assembles statement-2 if the expression is false (0 or no character string data).
- Assembles statement-1 if the expression that follows .IF is true; assembles statement-2 if the expression is false.
- This instruction may be nested up to 20 levels.
- Multiple lines may be specified for statement-1 and -2.
- Label or symbol used in expression must be defined before this line.
- Label with relocatable value may not be specified in the operand field.
- The operand may contain the following logical instructions.

**Table B.1 Allowed Logical Instructions**

<	Less than
>	Greater than
==	Equal
!=	Not equal
<=	Less than or equal
>=	Greater than or equal

### Example:

(1)

```
IF FLAG ; Assembles lines through .ELSE if FLAG is true.
:
:
.ELSE; Assembles lines through .ENDIF if FLAG is false.
:
:
.ENDIF
```

(2)

```
ADD: .MACRO      OP1,OP2,OP3
      .IF        "OP3"      ; Assemble through .ELSE if argument
                          ; OP3 exists
      ADC        OP1,OP2,OP3
      .ELSE
      ADC        OP1,OP2
      .ENDIF
```

### **.INCLUDE**

Include a file

---

**Format:**

▲. INCLUDE△file-name

**Description:**

- Includes the file specified by the operand where this pseudo instruction is specified.
- The full filename must be specified for file-name.
- This pseudo instruction may be nested up to 9 levels.
- The nesting level is output on the print file.

**Example:**

```
.INCLUDE TEST.INC ; Include contents of TEST.INC file.
```

### **.INDEX**

Declare index register length (default is 16)

---

**Format:**

▲.INDEX△expression

**Description:**

- Declares the length (8 or 16) of CPU internal index register.
- The length is 8 bit if value of expression is 8; 16 bit if value of expression is 16.
- The new index register length must be declared with this pseudo instruction when changing the index register length selection flag with the “CLP X” or “SEP X” instruction.
- Index register length can be specified by adding the prefix “.B” or “.W” to op-code. If the length specified by op-code differs from that specified by this pseudo instruction, the program is assembled using the op-code specified length.
- Note that this pseudo instruction only declares the index register length for the assembler, and it does not manipulate the index register length selection flag (x) for the CPU internal processor status register.

**Example:**

```
SEP      X           ; Sets X flag.  
.INDEX   8           ; Specifies index register length.  
LDX     #DATA       ; Executes 8 bit load.
```

---

**.LIB**Specify library filename

---

**Format:**

▲.LIB△file-name[,file-name,...,file-name]

**Description:**

- Specifies the names of the library files to be linked.
- Only files with extension .LIB may be specified. Linkage error will occur if other files are specified.
- This pseudo instruction may not be placed in a nest.
- Directory path and file extension (.LIB) may not be specified for the filenames.

**Example:**

```
.LIB LIB1,LIB2,LIB3
```

---

**.LINE**Specify lines per page (default is 54)

---

**Format:**

▲.LINE△expression

**Description:**

- Specifies the number of lines (5-255) per-listing page.
- This pseudo instruction may be specified only once in a program.

**Example:**

```
.LINE60 ; Sets 60 lines per page.
```

### **.LIST**

Start list output (default)

---

**Format:**

▲.LIST

**Description:**

- Outputs list to a PRN file.
- This pseudo instruction is used to resume list output to PRN file after it has been interrupted by .NLIST.

**Example:**

```
.NLIST      ; Suppresses listing output.  
:          ; No output to PRN file through "LIST".  
:  
.LIST      ; Start listing output.  
:          ; Output subsequent lines to PRN file.
```

### **.LISTM**

Start list output of macro expansion

---

**Format:**

▲.LISTM

**Description:**

- Outputs macro expansion list to a PRN file.

**Note:**

- The .LISTM instruction is ignored if the entire list output is suppressed with the .NLIST instruction.
- The .LISTM instruction is ignored if the command parameter "-M" is not specified.

**Example:**

```
.NLISTM     ; Suppresses list output of macro expansion.  
:          ; List output of macro expansion is suppressed  
:          ; until ".LISTM".  
.LISTM     ; Starts list output of macro expansion.  
:          ; Macro expansions are output after this instruction.
```



---

**.MCU****Sets instruction generating MCU type**

---

**Format:****▲.MCU△MCU Type****Description:**

- Sets the MCU type for which instructions are generated.
- The following types can be written in the operand. (Types can be written in either uppercase or lowercase letters.)
  1. M37700  
Generates the 7700-family instructions.
  2. M37750  
Generates the 7750-series instructions.
  3. M37751  
Generates the 7751-series instructions.
- The MCU type set by this pseudo-instruction remains effective until another type is set by ".MCU."
- If this pseudo-instruction is omitted, the assembler generates the 7700-family instructions. (An error result if the written mnemonic for M37750 or M37751.)

**Example:**

```
.MCU      M37750      ;Specifies generation of the M37750 instructions.
```

### **.NLIST**

Suppress listing output

---

**Format:**

▲.NLIST

**Description:**

- Suppresses output to PRN file.
- List output can be resumed with the pseudo instruction LIST.

**Example:**

```
.NLIST      ; Suppresses listing output.  
:          ; Suppress output to PRN file until "LIST".  
:  
.LIST      ; Start list output.  
:          ; Output subsequent lines to PRN file.  
:
```

### **.NLISTM**

Suppress macro expansion list

---

**Format:**

▲.NLISTM

**Description:**

- Suppresses output of macro expansion list to PRN file.
- List output can be resumed with the pseudo instruction LISTM.

**Note:**

- The .LISTM instruction is ignored if the command parameter "-M" is not specified.

**Example:**

```
.NLISTM    ; Suppresses list output of macro expansion.  
:          ; List output of macro expansion is suppressed  
:          ; until ".LISTM".  
.LISTM     ; Starts list output of macro expansion.  
:          ; Macro expansions are output after this instruction.
```

---

**.OBJ****Specify relocatable filename**

---

**Format:**

▲.OBJ△file-name[,file-name,...,file-name]

**Description:**

- Specifies the names of relocatable files to be linked.
- Only the files with extension .R77 may be specified. Linkage error will occur if other files are specified.
- This pseudo instruction may not be placed in a nest.
- Directory path and file extension (.R77) may not be specified for the filenames.

**Example:**

```
.OBJ OBJ1,OBJ2,OBJ3
```

---

**.ORG****Declare location (default is 000000H)**

---

**Format:**

▲.ORG△expression

**Description:**

- Declares the starting address for the lines that follow this line.
- If this instruction is not specified, the assembler assumes 000000H as the starting address.
- Specification of this pseudo instruction results in assignment of the absolute attribute for the section. A section without a specification of this pseudo instruction is relocatable.
- Label or symbol used in expression must be defined before this line.
- Label with relocatable value may not be specified in the operand field.

**Example:**

```
.ORG 0C000H ; Sets location as 0C000H.
```

### **.PAGE**

Specify new page and title for listing

---

**Format:**

▲.PAGE△['title']

**Description:**

- Skips to new page immediately before this instruction, and outputs the title specified in the operand field in the list's header section. Title must be enclosed between single quotes ( ' ') or double quotes ( " ").
- The maximum number of characters permitted in title is 20 if column specification is 80, 45 if column specification is from 105 to 132, or 60 subtracted from the number of columns if column specification is 81 to 104. If title is not specified, only skip to new page is performed.

**Example:**

```
PAGE 'PROG1' ; Outputs PROG1 to the PRN file header
```

### **.PUB**

Declare public label or symbol

---

**Format:**

▲.PUB△label or symbol[,label or symbol,...,label or symbol]

**Description:**

- Declares that the symbols or labels specified in the operand field can be referenced from other source files.
- This pseudo instruction must be specified before the lines that define the labels or symbols.

**Example:**

```
                .PUB WORK1,WORK2,WORK3  
WORK1:         .BLKW 1  
WORK2:         .BLKW 1  
WORK3:         .BLKW 1
```

**.SECTION**

Declare section name

**Format:**

▲. SECTION△section-name

**Description:**

- RASM77 processes a source program in units called sections. This pseudo instruction declares the section name specified in the operand field as the name of the program section that begins on next line.
- Any name may be specified for section-name. There may be more than one section with same section name in a file.
- This pseudo instruction must always be specified before starting the program code. (An error occurs if it is not specified.)

**Example:**

```

        .SECTION  DATA ; DATA section begins.
datatop:
nulldt:      .BLKB 8
            :
        .SECTION  STACK ; STACK section begins.
        .BLKB1000H
stacktop:
            :
        .SECTION  PROG ; PROG section begins.
_init:
        .DATA16
        .INDEX   16
        CLP      #0FFH
        LDA      A,#stacktop
        TAS
        LDA      A,#SIZEOF DATA
        LDX      #OFFSET constop
        LDY      #OFFSET datatop
        MVN      BANK constop,BANK datatop
            :
        .SECTION  CONT ; CONT section begins.
constop:
        .BYTE0,0,0,0,0,0,0,0
            :

```

**.VER****Declare program version**

---

**Format:**

▲.VER△'character-string'

**Description:**

- Declares the program version for relocatable files.
- When the “-V” parameter is specified, LINK77 checks that relocatable files have the same version. This feature enables confirmation of version consistency between relocatable files that are being linked. For detailed explanation of the “- V” parameter, refer to “LINK77 Operation Manual”.
- Version consistency check is performed by character string comparison. Note that LINK77 recognizes uppercase and lowercase characters as different characters.
- This pseudo instruction may be specified only once in a program.

**Example:**

```
.VER 'V.1.0' ; Declares "V.1.0" as the program version.
```

**.WORD****Define word data**

---

**Format:**

A [label:] ▲.WORD△expression

**Description:**

- Defines the value of expression as a word data.
- Up to 16 data can be specified, but each data must be delimited by ‘,’.
- Data is defined from low-order byte (it is defined from the high-order byte in the OBJ field of the PRN file).
- Global label may be specified in the operand field.

**Example:**

```
label: .WORD      0E000H          ; Defines 00H and E0H.  
      .WORDOFFSET symbol        ; Defines value of symbol  
                                   ; from low-order byte.
```

## B.3 Debugging Pseudo Instructions

The following pseudo instructions are for source line debugging. These pseudo instructions pass source line debugging information of programs coded in 7700 Family C language and structured preprocessor to debugger. These pseudo instructions are generated by the preprocessor.

**.CLINE****Output line number information**

---

**Format:****▲.CLINE△numeric-value****Description:**

- Sets the source line information necessary during debugging.
- This pseudo instruction is output to assembly file compiled by C compiler or processed by PRE77.

**Notes:**

- A number from 1 to 9999 can be specified as the operand.
- If a value greater than 9999 is specified, a warning is issued and the line number information is not output to object file.
- The number specified in the operand is not checked for redefinition.
- This instruction is ignored if it is within a macro code.

**Example:**

```
.CLINE 10
JSR   _SUB
.CLINE 11
:
```

**.ENDFUNC****Specify end of function**

---

**Format:****▲.ENDFUNC△label****Description:**

- Specifies the end of function (subroutine).
- This instruction enables source line debugging.
- This pseudo instruction cannot be nested.

**Example:**

```
      .FUNC      SUB
SUB:   LDA      A,#0
      :
      RTS
      .ENDFUNC  SUB      ; Specifies the end of function
```



---

**.FUNC**Specify start of function

---

**Format:**

▲.FUNC△label

**Description:**

- Specifies the start of function (subroutine).
- This instruction enables source line debugging.
- This pseudo instruction cannot be nested.

**Example:**

```
          .FUNC      SUB      ; Specifies the start of function
SUB:     LDA        A,#0
          :
          RTS
          .ENDFUNC   SUB
```

---

**.LANGUAGE**Output language name

---

**Format:**

▲.LANGUAGE△language-name

**Description:**

- Sets the language information necessary for source debugging.

**Notes:**

- Language information is not output to object file if the command line parameter “-C” is not specified.
- This pseudo instruction is allowed only once in a source file.

**Example:**

```
          .LANGUAGE   C
```

### **.POINTER**

Define pointer length

---

**Format:**

▲.POINTER△numeric-value

**Description:**

- Defines the byte length of pointer variable used with C compiler.
- A number from 1 to 255 can be specified as operand.

**Note:**

- This pseudo instruction is allowed only once in a source file.

**Example:**

```
.POINTER 2
```

### **.SOURCE**

Define source file name

---

**Format:**

▲.SOURCE△source-file-name

**Description:**

- Defines the file name necessary during source debugging.

**Notes:**

- Uppercase and lowercase characters are distinguished in the file name.
- This pseudo instruction must be coded before the .CLINE instruction.

**Example:**

```
.SOURCE  PROG1.C  
.CLINE   1  
:  
.SOURCE  A:/INCLUDE/STDIO.H  
.CLINE   1
```

## **B.4 Reserved Pseudo Instructions**

The pseudo instructions described below are reserved for future expansion of RASM77. These pseudo instructions, even if specified, will not affect assembly.

Note:

Note that if only one of a combination of two pseudo-instructions ".IO - .ENDIO," ".PROCINT - .ENDPROC," ".PROCMAIN - .ENDPROC," ".PROCSUB - .ENDPROC," or ".RAM - .ENDRAM" is written or a combination of two pseudo-instructions is written in an incorrect sequence, RASM77 generates an assemble error.

### **.ENDIO**

Declares end of I/O area (Reserved)

---

**Format:**

▲.ENDIO

**Description:**

- Declares the end of an I/O area.
- Labels and symbols that are specified between .I/O and .ENDIO are interpreted as an I/O area.

**Example:**

```
.IO
port0 .EQU 00H
port1 .EQU 01H
.ENDIO
```

### **.ENDPROC**

Declares end of program module (Reserved)

---

**Format:**

▲.ENDPROC

**Description:**

- Declares the end of a main program module, a sub-program module, or an interrupt handler module.
- The lines between .PROCINT, .PROCMAIN or PROC SUB and .ENDPROC are interpreted as one program module.

**Example:**

```
.PROCMAIN
MAIN:
:
JMP MAIN
.ENDPROC
```

---

**.ENDRAM**Declares end of RAM area (Reserved)

---

**Format:**

▲.ENDRAM

**Description:**

- Declares the end of RAM area.
- The labels and symbols specified between RAM and .ENDRAM are interpreted as a RAM area.

**Example:**

```
        .RAM  
work0:  .BLKB    1  
work1:  .BLKB    1  
        .ENDRAM
```

---

**.IO**Declares start of I/O area (Reserved)

---

**Format:**

▲.IO

**Description:**

- Specifies start of an I/O area declaration.
- The labels and symbols specified between .IO and .ENDIO are interpreted as an I/O area.

**Example:**

```
        .IO  
port0 .EQU 00H  
port1 .EQU 01H  
        .ENDIO
```

### **.PROCINT**

Declares start of interrupt handler (Reserved)

---

**Format:**

▲.PROCINT△[label]

**Description:**

- Declares the beginning of interrupt handler.
- The lines between .PROCINT and .ENDPROC are interpreted as the interrupt handler.
- RASM77 processes the label specified in the operand field as the label for this line.

**Example:**

```
.PROCINT INT
:
:
.ENDPROC
```

### **.PROCMAIN**

Declares start of main program (Reserved)

---

**Format:**

▲.PROCMAIN△[label]

**Description:**

- Declares the beginning of a main program.
- The lines between .PROCMAIN and .ENDPROC are interpreted as the main program.
- RASM77 processes the label specified in the operand field as the label for this line.

**Example:**

```
.PROCMAIN MAIN
:
:
.ENDPROC
```

---

**.PROCSUB****Declare beginning of sub-program (Reserved)**

---

**Format:**

▲.PROCSUB△[label]

**Description:**

- Declares the beginning of a sub-program.
- The lines between .PROCSUB and .ENDPROC are interpreted as the subprogram.
- RASM77 processes the label specified in the operand field as the label for this line.

**Example:**

```
.PROCSUB  SUB
:
:
.ENDPROC
```

---

**.PROGNAME****Declares program name (Reserved)**

---

**Format:**

▲. PROGNAME△program-name

**Description:**

- Declares a program name.
- Specification in the operand field is interpreted as the program title.

**Example:**

```
.PROGNAME 'printer control program'
```

**.RAM**

Declares start of RAM area (Reserved)

---

**Format:**

▲.RAM

**Description:**

- Specifies start of declaration of a RAM area.
- The labels and symbols specified between .RAM and .ENDRAM are interpreted as the RAM area.

**Example:**

```
.RAM  
work0: .BLKB 1  
work1: .BLKB 1  
.ENDRAM
```



# APPENDIX C

## Macro Instructions

### C.1 Conventions

The macro instructions available with RASM77 are described in alphabetical order. The following conventions are used in describing each macro:

1. Item in [ ] may be omitted.
2. Space or tab code is indicated by  $\triangle$  or  $\blacktriangle$ .  $\triangle$  is a required space or tab code, and  $\blacktriangle$  is an optional space or tab code (i.e., may be omitted).
3.  $\blacktriangle$  is used to separate a label from macro instruction. When specifying a label, a colon (:) is not required but, if it is omitted, either a space or a tab code must be specified between the label and macro instruction.

### C.2 Macro Instructions

### **.ENDM**

Declares end of macro

---

**Format:**

▲.ENDM

**Description:**

- This instruction specifies the end of all macro definitions.

**Example:**

[Macro Definition]

```
ADD:  .MACRO  VAL
      CLC
      ADC    A, VAL
      .ENDM
```

[Macro Call]

```
ADD    #10
```

[Macro Expansion]

```
CLC
ADC    A, #10
.ENDM
```

---

**.EXITM**Exit from macro

---

**Format:**

▲.EXITM

**Description:**

- This instruction cancels macro expansion and passes control to the nearest .ENDM.

**Example:****[Macro Definition]**

```
DATA1: .MACRO  VAL
        .IF    LABEL
        .BYTE  VAL
        .EXITM
        .ENDIF
        .WORD  VAL
        .ENDM
```

**[Macro Call]**

```
LABEL .EQU 1
DATA1 10
```

**[Macro Expansion]**

```
.IF LABEL
.BYTE 10
.EXITM
.ENDIF
.ENDM
```

### **.LOCAL**

Defines macro local label

---

#### **Format:**

▲.LOCAL△label,[label,...,label]

#### **Description:**

- This instruction defines a label defined within a macro as a local label.
- Labels declared as local are assigned labels ..n (n is decimal) in the order of appearance during assembly. Therefore, the user must not use labels beginning with two periods (..).
- Local declaration must be made before a label is used.

#### **Example:**

##### [Macro Definition]

```
LOOP:  .MACRO
        .LOCAL  LOOP1
        LDA    A, #20
LOOP1:  DEC    A
        BNE   LOOP1
        .ENDM
```

##### [Macro Call]

```
LOOP
```

##### [Macro Expansion]

```
        LDA    A, #20
..0:    DEC    A
        BNE   ..0
        .ENDM
```

---

**.MACRO - .ENDM**Defines macro

---

**Format:**

▲macro-name▲.MACRO△[argument 1,argument 2,...,argument n]

**Description:**

- A macro definition starts with the line .MACRO and ends with the line .ENDM.
- The string assigned to the .MACRO line becomes the name of the macro definition.
- Macro definition can have arguments or no arguments. If arguments are used, the necessary arguments must be passed during macro call.
- When a macro call is made, the arguments are passed in the order of dummy arguments in the macro definition.
- Assembly language instructions, user macros, system macros, and pseudo instructions other than .INCLUDE can be coded between .MACRO and .ENDM. Macro definitions can be nested up to 20 levels.
- Macro calls can be made anywhere in the program as long as it is after the macro definition. An error will occur if a macro call is issued before it is defined. Macro calls can be nested up to 20 levels.
- The arguments specified in the macro call operand are replaced with the dummy arguments in the macro definition from left to right. The number of arguments in the macro call and macro definition need not be the same, but a warning is issued. If the number of arguments on macro call is greater than the number of dummy arguments, the excess arguments are ignored. If the number of arguments on macro call is less than the number of dummy arguments, null character (string with length 0) is assigned to dummy arguments that have no corresponding actual argument.
- Any number of arguments can be specified on macro call regardless of the number of dummy arguments in the macro definition. However, all arguments must fit in one line. Each argument must be delimited by a comma. To pass a comma or space as argument, enclose it in double quotes. Commas inside parentheses are not treated as delimiters.
- Macro expansion lines are indicated in the list file with a plus sign.
- The same macro name can be used to define more than one macro. In this case, the definition immediately prior to the call takes effect when a call is made.

## APPENDIX C. MACRO INSTRUCTIONS

---

### Example:

#### Example 1) Macro definition without operand

##### [Macro Definition]

```
ADD1:  .MACRO
        LDA    A,ABC
        LDX    #DEF
        CLC
        ADC    A, TABLE, X
        STA    A, GHI
        .ENDM
```

##### [Macro Call]

```
ADD1
```

##### [Macro Expansion]

```
LDA    A,ABC
LDX    #DEF
CLC
ADC    A, TABLE, X
STA    A, GHI
.ENDM
```

#### Example 2) Macro definition with operand

##### [Macro Definition]

```
ADD2:  .MACRO  V1, IMM, V2
        ↑
        Dummy argument
        LDA    A, V1
        LDX    #IMM
        CLC
        ADC    A, TABLE, X
        STA    A, V2
        .ENDM
```

##### [Macro Call]

```
ADD2    WORK1, 10, WORK2
```

##### [Macro Expansion]

```
LDA    A, WORK1
LDX    #10
CLC
ADC    A, TABLE, X
STA    A, WORK2
.ENDM
```

### Example:

#### Example 3) Macro nesting

##### [Macro Definition]

```
ADD:      .MACRO  OP1,OP2
          CLC
          ADC     OP1,OP2
          .ENDM
```

##### [Macro Call]

```
ADD2:     .MACRO  OP1,OP2
          ADD     OP1,OP2      ; Macro call within a macro
          ADC     OP1+1,OP2+2
          .ENDM
```

#### Example 4) Recursive definition

##### [Macro Definition]

```
MAC:      .MACRO                ; First definition
DATA:     .BLKB  1
MAC:      .MACRO  VALUE         ; Second definition
LDM       #VALUE,DATA
          .ENDM
          .ENDM
```

##### [Macro Call]

```
MAC                ; Reserve area and define new macro
:
MAC  10H           ; Call newly defined macro
```

**Format:**

▲[label:]▲.REPEAT△count

**Description:**

- Repeats assembly of 7700 Family instructions between .REPEAT and .ENDM for the number of times specified with the operand.
- The instruction can be repeated up to 254 times.
- The label of the .REPEAT instruction is assigned as the label of the first generated line.
- Assembly language instructions, user macros, system macros, and pseudo instructions other than .INCLUDE can be coded between .REPEAT and .ENDM.
- The operand may be a numeric constant or a symbol constant (label), a label with relocatable value cannot be specified.
- Nesting is allowed up to 20 levels.

**Example:****[Source Code Example]**

```
TIME5:  .REPEAT  5
        NOP
        .ENDM
```

**[After Macro Expansion]**

```
TIME5:  .REPEAT  5
        NOP
        .ENDM

TIME5:
        NOP
        NOP
        NOP
        NOP
        NOP
        .ENDM
```



---

**.REPEATC - .ENDM**Defines REPEATC macro

---

**Format:**

▲[label:]▲.REPEATC△dummy-argument,actual-argument

**Description:**

- Repeats assembly of statements up to .ENDM for the number of times specified with the actual argument.
- One character is extracted from actual argument and passed to the dummy argument each time.
- The label of the .REPEATC instruction is assigned as the label of the first generated line.
- Assembly language instructions, user macros, system macros, and pseudo instructions other than .INCLUDE can be coded between .REPEATC and .ENDM.
- If the character string contains special characters such as space, tab, or comma, the entire string must be enclosed in double quotes. In this case, the string with the quotes removed is used.
- Nesting is allowed up to 20 levels.

## APPENDIX C. MACRO INSTRUCTIONS

---

### Example:

#### Example 1)

##### [Source Code Example]

```
DATA:   .REPEATC VAL,ABCDE
        ↑      ↑
```

Dummy argument Actual argument

```
.BYTE  'VAL'
.ENDM
```

##### [After Macro Expansion]

```
DATA:   .REPEATC VAL,ABCDE
        .BYTE  'VAL'
        .ENDM
```

```
DATA:
        .BYTE  'A'
        .BYTE  'B'
        .BYTE  'C'
        .BYTE  'D'
        .BYTE  'E'
        .ENDM
```

#### Example 2)

##### [Source Code Example]

```
DATA:   .REPEATC VAL,"ABC,;"
        ↑      ↑
```

Dummy argument Actual argument

```
.BYTE  'VAL'
.ENDM
```

##### [After Macro Expansion]

```
DATA:   .REPEATC VAL,"ABC,;"
        .BYTE  'VAL'
```

```
DATA:
        .BYTE  'A'
        .BYTE  'B'
        .BYTE  'C'
        .BYTE  ','
        .BYTE  ';'
        .ENDM
```

---

**.REPEATI - .ENDM****Defines REPEATI macro**

---

**Format:**

▲[label:]▲.REPEATI△dummy-argument,actual-argument[,dummy-argument,..actual-argument]

**Description:**

- Repeats assembly of statements up to .ENDM for the number of actual argument specified in the operand.
- One actual argument is extracted and passed to the dummy argument each time.
- The label of the .REPEATI instruction is assigned as the label of the first generated line.
- Assembly language instructions, user macros, system macros, and pseudo instructions other than .INCLUDE can be coded between .REPEATI and .ENDM.
- Numeric constant, character constant, symbol constant (label), and character string can be specified for actual argument. Other macro instructions cannot be specified.
- If the actual argument contains special characters such as space, tab, or comma, the entire string must be enclosed in double quotes. In this case, the string with the quotes removed is used.
- Nesting is allowed up to 20 levels.

## APPENDIX C. MACRO INSTRUCTIONS

---

### Example:

#### Example 1)

##### [Source Code Example]

```
SUB:  .REPEATI INST,"NOP","LDA A,#1","JSR SUB1","RTS"
      ↑           ↑
      Dummy argument Actual argument
      INST
      .ENDM
```

##### [After Macro Expansion]

```
SUB:  .REPEATI INST,"NOP","LDA A,#1","JSR SUB1","RTS"
      INST
      .ENDM

SUB:
      NOP
      LDA A,#1
      JSR SUB1
      RTS
      .ENDM
```

#### Example 2)

##### [Source Code Example]

```
DATA:  .REPEATI VAL,0,1,2,"HELLO !!!"
      ↑           ↑
      Dummy argument Actual argument
      .BYTE VAL
      .ENDM
```

##### [After Macro Expansion]

```
DATA:  .REPEATI VAL,0,1,2,"HELLO !!!"
      .BYTE VAL
      .ENDM

DATA:
      .BYTE 0
      .BYTE 1
      .BYTE 2
      .BYTE 'HELLO !!!'
      .ENDM
```

# APPENDIX D

## Instruction Set

### D.1 Symbols

Table D.1 lists the meaning of symbols used in the list of instructions.

**Table D.1 Symbols for Instruction List**

Acc	Accumulator A or accumulator B
X	Index register X
Y	Index register Y
S	Stack pointer S
d8	8-bit data
d16	16-bit data
[DP:]zz	8-bit relative address from direct page register in direct addressing mode
[Dt:]hhll	Low-order 16-bit address in absolute addressing mode
[LG:]hhmlll	24-bit address in absolute long addressing mode
imm	8-bit and 16-bit immediate data
rr	Relative address from -128 to +127
rlll	Relative address from -65535 to +65534
DPR	Direct page register
PG	Program bank register
DT	Data bank register
PS	Processor status register
C	Carry flag
Z	Zero flag
I	Interrupt disable flag
D	Decimal mode flag
X	Index register length selection flag
M	Data length selection flag
V	Overflow flag
N	Negative flag
DBR	Data bank register
PBR	Program bank register
PSR	Processor status register

Notes:

1. [DP:], [DT:], [LG:]
  - Use these symbols to explicitly specify direct addressing, absolute addressing, or absolute long addressing.
  - zz or hlll is treated as offset from the beginning of the direct page or data bank register only when symbol is specified (including symbol specified with the “-D” parameter).
  - If label is specified for zz or hlll, the difference between the value of the label and the value in the currently specified register is treated as offset.
2. Immediate (imm)
  - Normally, the data length is the default length defined with the .DATA or .INDEX pseudo instruction. However, it can be explicitly specified for each instruction by appending the symbol “.B” or “.W” after the instruction code.

Example 1:    ADC.B            A, #data  
              => Assembled as 8-bit data.

Example 2:    ADC.W        A, #data  
              => Assembled as 16-bit data.

- RASM77 does not change the status of the CPU internal data length selection flag (m) and index register length selection flag (x) . This is left up to the user.
3. The ‘X’ in the operand of the CLP or SEP instruction is assumed to be the index register length selection flag.

## D.2 Instruction Set

Table D.2 shows all of the instructions available with RASM77. The allowed data length specification, addressing mode name, and coding format are shown next to each instruction.

## APPENDIX D. INSTRUCTION SET

Table D.2 Instructions

Instruction	Data Length	Addressing Mode	Coding Format
ADC	.B/.W	Immediate	ADC.B Acc,#imm
ADCL		Direct	ADC Acc,[DP:]zz
		Direct X	ADC Acc,[DP:]zz,X
		Direct indirect	ADC Acc,([DP:]zz)
		Direct indirect X	ADC Acc,([DP:]zz,X)
		Direct indirect Y	ADC Acc,((DP:]zz),Y
		Direct indirect long	ADCL Acc,([DP:]zz)
		Direct indirect long Y	ADCL Acc,([DP:]zz),Y
		Absolute	ADC Acc,(DT:]hhll
		Absolute X	ADC Acc,[DT:]hhll,X
		Absolute Y	ADC Acc,[DT:]hhll,Y
		Absolute long	ADC Acc,[LG:]hhmmll
		Absolute long X	ADC Acc,(LG:]hhmmll,X
		Stack pointer relative	ADC Acc,d8,S
		Stack pointer relative indirect Y	ADC Acc,(d8,S),Y
AND	.B / .W	Immediate	AND.B Acc,#imm
ANDL		Direct	AND Acc,[DP:]zz
		Direct X	AND Acc,[DP:]zz,X
		Direct indirect	AND Acc,([DP:]zz)
		Direct indirect X	AND Acc,([DP:]zz,X)
		Direct indirect Y	AND Acc,([DP:]zz),Y
		Direct indirect long	ANDL Acc,([DP:]zz)
		Direct indirect long Y	ANDL Acc,([DP:]zz),Y
		Absolute	AND Acc,[DT:]hhll
		Absolute X	AND Acc,[DT:]hhll,X
		Absolute Y	AND Acc,[DT:]hhll,Y
		Absolute long	AND Acc,[LG:]hhmmll
		Absolute long X	AND Acc,[LG:]hhmmll,X
		Stack pointer relative	AND Acc,d8,S
		Stack pointer relative indirect Y	AND Acc,(d8,S),Y



Instruction	Data Length	Addressing Mode	Coding Format
ASL		Accumulator	ASL Acc
		Direct	ASL [DP:]zz
		Direct X	ASL [DP:]zz,X
		Absolute	ASL [DT:]hhll
		Absolute X	ASL [DT:]hhll,X
ASR		Accumulator	ASR Acc
		Direct	ASR [DP:]zz
		Direct X	ASR [DP:]zz,X
		Absolute	ASR [DT:]hhll
		Absolute X	ASR [DT:]hhll,X
BBC	.B/.W	Direct Bit Relative	BBC.B #imm,[DP:]ZZ,rr
	.B/.W	Absolute Bit Relative	BBC.B #imm,[DT:]hhll,rr
BBS	.B/.W	Direct Bit Relative	BBS.B #imm,[DP:]zz,rt
	.B/.W	Absolute Bit Relative	BBS.B #imm,[DT:]hhll,rr
BCC		Relative	BCC rr
BCS		Relative	BCS rr
BEQ		Relative	BEQ rr
BMI		Relative	BMI rr
BNE		Relative	BNE rr
BPL		Relative	BPL rr
BRA		Relative	BRA rr
BRAL		Relative	BRAL rrl
BRK		Implied	BRK #d8
BVC		Relative	BCC rr
BVS		Relative	BCS rr
CLB	.B/.W	Direct Bit	CLB.B #imm,[DP:]zz
	.B/.W	Absolute Bit	CLB.B #imm,[DT:]hhll
CLC		Implied	CLC
CLI		Implied	CLI
CLM		Implied	CLM
CLP <sup>1</sup>		Immediate	CLP #d8
CLV		Implied	CLV

Note:

1. The CLP instruction can have register names in the operand field.

Example: `CLP C,Z,I,D,X,M,V,N`

## APPENDIX D. INSTRUCTION SET

Instruction	Data Length	Addressing Mode	Coding Format
CMP	.B/.W	Immediate	CMP.B Acc,#imm
CMPL		Direct	CMP Acc,[DP:]zz
		Direct X	CMP Acc,[DP:]zz,X
		Direct indirect	CMP Acc,([DP:]zz)
		Direct indirect X	CMP Acc,([DP:]zz,X)
		Direct indirect Y	CMP Acc,([DP:]zz),Y
		Direct indirect long	CMPL Acc,([DP:]zz)
		Direct indirect long Y	CMPL Acc,([DP:]zz,Y
		Absolute	CMP Acc,[DT:]hhll
		Absolute X	CMP Acc,[DT:]hhll,X
		Absolute Y	CMP Acc,[DT:]hhll,Y
		Absolute long	CMP Acc,(LG:]hhmmlI
		Absolute long X	CMP Acc,[LG:]hhmmlI,X
		Stack pointer relative	CMP Acc,d8,S
		Stack pointer relative indirect Y	CMP Acc,(d8,S),Y
CPX	.B/.W	Immediate	CPX.B #imm
		Direct	CPX [DP:]zz
		Absolute	CPX [DT:] hhll
CPY	.B/.W	Immediate	CPY.B #imm
		Direct	CPY [DP:]zz
		Absolute	CPY [DT:]hhll
DEC		Accumulator	DEC Acc
		Direct	DEC [DP:]zz
		Direct X	DEC [DP:]zz,X
		Absolute	DEC [DT:]hhll
		Absolute X	DEC [DT:]hhll,X
DEX		Implied	DEX
DEY		Implied	DEY

Instruction	Data Length	Addressing Mode	Coding Format
DIV	.B/.W	Immediate	DIV.B #imm
DIVL		Direct	DIV [DP:]zz
		Direct X	DIV [DP:]zz,X
		Direct indirect	DIV ([DP:]zz)
		Direct indirect X	DIV ([DP:]zz,X)
		Direct indirect Y	DIV ([DP:]zz),Y
		Direct indirect long	DIVL ([DP:]zz)
		Direct indirect long Y	DIVL ([DP:]zz),Y
		Absolute	DIV [DT:]hhll
		Absolute X	DIV [DT:]hhll,X
		Absolute Y	DIV [DT:]hhll,Y
		Absolute long	DIV [LG:]hhmml
		Absolute long X	DIV [LG:]hhmml,X
		Stack pointer relative	DIV d8,S
		Stack pointer relative indirect Y	DIV (d8,s),Y
DIVS	.B/.W	Immediate	DIVS.B #imm
DIVSL		Direct	DIVS [DP:]zz
		Direct X	DIVS [DP:]zz,X
		Direct indirect	DIVS ([DP:]zz)
		Direct indirect X	DIVS ([DP:]zz,X)
		Direct indirect Y	DIVS ([DP:]zz),Y
		Direct indirect long	DIVSL ([DP:]zz)
		Direct indirect long Y	DIVSL ([DP:]zz),Y
		Absolute	DIVS [DT:]hhll
		Absolute X	DIVS [DT:]hhll,X
		Absolute Y	DIVS [DT:]hhll,Y
		Absolute long	DIVS [LG:]hhmml
		Absolute long X	DIVS [LG:]hhmml,X
		Stack pointer relative	DIVS d8,S
		Stack pointer relative indirect Y	DIVS (d8,s),Y

## APPENDIX D. INSTRUCTION SET

Instruction	Data Length	Addressing Mode	Coding Format
EOR	.B/.W	Immediate	EOR.B Acc,#imm
EORL		Direct	EOR Acc,[DP:]zz
		Direct X	EOR Acc,[DP:]zz,X
		Direct indirect	EOR Acc,([DP:]zz)
		Direct indirect X	EOR Acc,([DP:]zz,X)
		Direct indirect Y	EOR Acc,([DP:]zz),Y
		Direct indirect long	EORL Acc,([DP:]zz)
		Direct indirect long Y	EORL Acc,([DP:]zz),Y
		Absolute	EOR Acc,[DT:]hhll
		Absolute X	EOR Acc,[DT:]hhll,X
		Absolute Y	EOR Acc,[DT:]hhll,Y
		Absolute long	EOR Acc,[LG:]hhmml
		Absolute long X	EOR Acc,[LG:]hhmml,X
		Stack pointer relative	EOR Acc,d8,S
		Stack pointer relative indirect Y	EOR Acc(d8,S),Y
EXTS		Accumulator	EXTS Acc
EXTZ		Accumulator	EXTZ Acc

Instruction	Data Length	Addressing Mode	Coding Format
INC		Accumulator	INC    Acc
		Direct	INC    [DP:]zz
		Direct X	INC    [DP:]zz,X
		Absolute	INC    [DT:]hhll
		Absolute X	INC    [DT:]hhll,X
INX		Implied	INX
INY		Implied	INY
JMP JMPL		Absolute	JMP    hhll
		Absolute long	JMPL   (LG:)hhmml
		Absolute indirect	JMP    (hhll)
		Absolute Indirect Long	JMPL   (hhll)
		Absolute Indirect Indexed X	JMP    (hhll,x)
JSR JSRL		Absolute	JSR    hhll
		Absolute long	JSRL   [LG:]hhmml
		Absolute indirect X	JSR    (hhll,x)
LDA LDAL	.B/.W	Immediate	LDA.B   Acc,#imm
		Direct	LDA    Acc,[DP:]zz
		Direct X	LDA    Acc,(DP:]zz,X
		Direct indirect	LDA    Acc,([DP:]zz)
		Direct indirect X	LDA    Acc,([DP:]zz,X)
		Direct indirect Y	LDA    Acc,([DP:]zz),Y
		Direct indirect long	LDAL   Acc,([DP:]zz)
		Direct indirect long Y	LDAL   Acc,([DP:]zz),Y
		Absolute	LDA    Acc,[DT:]hhll
		Absolute X	LDA    Acc,[DT:]hhll,X
		Absolute Y	LDA    Acc,[DT:]hhll,Y
		Absolute long	LDA    Acc,[LG:]hhmml
		Absolute long X	LDA    Acc,(LG:]hhmml,X
		Stack pointer relative	LDA    Acc,d8,S
		Stack pointer relative indirect Y	LDA    Acc,(d8,S),Y
LDT		Immediate	LDT    #d8

## APPENDIX D. INSTRUCTION SET

Instruction	Data Length	Addressing Mode	Coding Format
LDM	.B/.W	Direct	LDM.B #imm,[DP:]zz
	.B/.W	Direct X	LDM.B #imm,[DP:]zz,X
	.B/.W	Absolute	LDM.B #imm,[DT:]hhll
	.B/.W	Absolute X	LDM.B #imm,[DT:]hhll,X
LDX	.B/.W	Immediate	LDX.B #imm
		Direct	LDX [DP:]zz
		Direct Y	LDX [DP:]zz,Y
		Absolute	LDX [DT:]hhll
		Absolute Y	LDX [DT:]hhll,Y
LDY	.B/.W	Immediate	LDY.B #imm
		Direct	LDY [DP:]zz
		Direct X	LDY [DP:]zz,X
		Absolute	LDY [DT:]hhll
		Absolute X	LDY [DT:]hhll,X
LSR		Accumulator	LSR Acc
		Direct	LSR [DP:]zz
		Direct X	LSR [DP:]zz,X
		Absolute	LSR [DT:]hhll
		Absolute X	LSR [DT:]hhll,X

Instruction	Data Length	Addressing Mode	Coding Format
MPY	.B/.W	Immediate	MPY.B #imm
MPYL		Direct	MPY [DP:]zz
		Direct X	MPY [DP:]zz,X
		Direct indirect	MPY ([DP:]zz)
		Direct indirect X	MPY ([DP:]zz,X)
		Direct indirect Y	MPY ([DP:]zz),Y
		Direct indirect long	MPYL ([DP:]zz)
		Direct indirect long Y	MPYL ([DP:]zz),Y
		Absolute	MPY [DT:]hhll
		Absolute X	MPY [DT:]hhll,X
		Absolute Y	MPY [DT:]hhll,Y
		Absolute long	MPY [LG:]hhmml
		Absolute long X	MPY [LG:]hhmml,X
		Stack pointer relative	MPY d8,S
		Stack pointer relative indirect Y	MPY (d8,S),Y
MPYS	.B/.W	Immediate	MPYS.B #imm
MPYSL		Direct	MPYS [DP:]zz
		Direct X	MPYS [DP:]zz,X
		Direct indirect	MPYS ([DP:]zz)
		Direct indirect X	MPYS ([DP:]zz,X)
		Direct indirect Y	MPYS ([DP:]zz),Y
		Direct indirect long	MPYSL ([DP:]zz)
		Direct indirect long Y	MPYSL ([DP:]zz),Y
		Absolute	MPYS [DT:]hhll
		Absolute X	MPYS [DT:]hhll,X
		Absolute Y	MPYS [DT:]hhll,Y
		Absolute long	MPYS [LG:]hhmml
		Absolute long X	MPYS [LG:]hhmml,X
		Stack pointer relative	MPYS d8,S
		Stack pointer relative indirect Y	MPYS (d8,S),Y

## APPENDIX D. INSTRUCTION SET

Instruction	Data Length	Addressing Mode	Coding Format
MVN		Block Transfer	MVN d8,d8
MVP		Block Transfer	MVP d8,d8
NOP		Implied	NOP
ORA	.B/.W	Immediate	ORA.B Acc,#imm
ORAL		Direct	ORA Acc,[DP:]zz
		Direct X	ORA Acc,[DP:]zz,X
		Direct indirect	ORA Acc,([DP:]zz)
		Direct indirect X	ORA Acc,([DP:]zz,X)
		Direct indirect Y	ORA Acc,([DP:]zz),Y
		Direct indirect long	ORAL Acc,([DP:]zz)
		Direct indirect long Y	ORAL Acc,([DP:]zz),Y
		Absolute	ORA Acc,[DT:]hhll
		Absolute X	ORA Acc,[DT:]hhll,X
		Absolute Y	ORA Acc,[DT:]hhll,Y
		Absolute long	ORA Acc,(LG:]hhmml
		Absolute long X	ORA Acc,[LG:]hhmml,X
		Stack pointer relative	ORA Acc,d8,S
		Stack pointer relative indirect Y	ORA Acc,(d8,S),Y
PEA		Stack	PEA #d16
PEI		Stack	PEI #d8
PER		Stack	PER #d16
PHA		Stack	PHA
PHB		Stack	PHB
PHD		Stack	PED
PHG		Stack	PHG
PHP		Stack	PEP
PHT		Stack	PHT
PHX		Stack	PEX
PHY		Stack	PHY



Instruction	Data Length	Addressing Mode	Coding Format
PLA		Stack	PLA
PLB		Stack	PLB
PLD		Stack	PLD
PLP		Stack	PLP
PLT		Stack	PLT
PLX		Stack	PLX
PLY		Stack	PLY
PSH <sup>1</sup>		Stack	PSH #d8
PUL <sup>2</sup>		Stack	PUL #d8
RLA	.B/.W	Immediate	RLA #imm
RMPA		Immediate	RMPA #imm
ROL		Accumulator	ROL Acc
		Direct	ROL [DP:]zz
		Direct X	ROL [DP:]zz,X
		Absolute	ROL [DT:]hhll
		Absolute X	ROL [DT:]hhll,X
ROR		Accumulator	ROR Acc
		Direct	ROR [DP:]zz
		Direct X	ROR [DP:]zz,X
		Absolute	ROR [DT:]hhll
		Absolute X	ROR [DT:]hhll,X
RTI		Implied	RTI
RTL		Implied	RTL
RTS		Implied	RTS

## Note:

- 1.,2. The PSH and PUL instruction can have register names in the operand field.

Example 1: PSH A,B,X,Y,DPR, PG,DT,PS

Example 2: PUL A,B,X,Y,DPR, PG,DT,PS

## APPENDIX D. INSTRUCTION SET

Instruction	Data Length	Addressing Mode	Coding Format
SBC	.B/.W	Immediate	SBC.B Acc,#imm
SBCL		Direct	SBC Acc,[DP:]zz
		Direct X	SBC Acc,[DP:]zz,X
		Direct indirect	SBC Acc,([DP:]zz)
		Direct indirect X	SBC Acc,([DP:]zz,X)
		Direct indirect Y	SBC Acc,([DP:]zz),Y
		Direct indirect long	SBCL Acc,([DP:]zz)
		Direct indirect long Y	SBCL Acc,([DP:]zz),Y
		Absolute	SBC Acc,[DT:]hhll
		Absolute X	SBC Acc,[DT:]hhll,X
		Absolute Y	SBC Acc,[DT:]hhll,Y
		Absolute long	SBC Acc,[LG:]hhmmlI
		Absolute long X	SBC Acc,[LG:]hhmmlI,:
		Stack pointer relative	SBC Acc,d8,S
		Stack pointer relative indirect Y	SBC Acc,(d8,S),Y
SEB	.B/.W	Direct Bit	SEB.B #imm,[DP:]zz
	.B/.W	Absolute Bit	SEB.B #imm,[DT:]hhll
SEC		Implied	SEC
SEI		Implied	SEI
SEM		Implied	SEM
SEP <sup>1</sup>		Immediate	SEP #d8

Note:

1. The SEP instruction can have register names in the operand field.

Example: `SEP C,Z,I,D,X,M,V,N`

Instruction	Data Length	Addressing Mode	Coding Format
STA		Direct	STA Acc,[DP:]zz
STAL		Direct X	STA Acc,[DP:]zz,X
		Direct indirect	STA Acc,([DP:]zz)
		Direct indirect X	STA Acc,([DP:]zz,X)
		Direct indirect Y	STA Acc,((DP:]zz),Y
		Direct indirect long	STAL Acc,([DP:]zz)
		Direct indirect long Y	STAL Acc,([DP:]zz),Y
		Absolute	STA Acc,[DT:]hhll
		Absolute X	STA Acc,[DT:]hhll,X
		Absolute Y	STA Acc,[DT:]hhll,Y
		Absolute long	STA Acc,hhmll
		Absolute long X	STA Acc,hhmll,X
		Stack pointer relative	STA Acc,d8,S
		Stack pointer relative indirect Y	STA Acc,(d8,S),Y
	STP		Implied
STX		Direct	STX [DP:]zz
		Direct Y	STX [DP:]zz,Y
		Absolute	STX [DT:]hhll
STY		Direct	STY [DP:]zz
		Direct X	STY [DP:]zz,X
		Absolute	STY [DT:]hhll
TAD		Implied	TAD
TAS		Implied	TAS
TAX		Implied	TAX
TAY		Implied	TAY
TBD		Implied	TBD
TBS		Implied	TBS
TBX		Implied	TBX
TBY		Implied	TBY

## APPENDIX D. INSTRUCTION SET

---

<b>Instruction</b>	<b>Data Length</b>	<b>Addressing Mode</b>	<b>Coding Format</b>
TDA		Implied	TDA
TDB		Implied	TDB
TSA		Implied	TSA
TSB		Implied	TSB
TSX		Implied	TSX
TXA		Implied	TXA
TXB		Implied	TXB
TXS		Implied	TXS
TXY		Implied	TXY
TYA		Implied	TYA
TYB		Implied	TYB
TYX		Implied	TYX
WIT		Implied	WIT
XAB		Implied	XAB

# APPENDIX E

## Instruction by Addressing Mode

### E.1 Instruction by Addressing Mode

The coding format and instructions for each addressing mode are shown below. The symbols are the same as those used in Appendix B.1.

#### 1. Implied

Instruction	Coding format
BRK, CLC, CLI, CLM, CLV DEX, DEY, INX, INY, NOP SEC, SEI, SEM, STP, TAX RTI, RTL, RTS, TAY, TAD TAS, TBB, TBD, TBS, TBX TBY, TDA, TDB, TSA, TSB TSX, TXA, TXB, TXS, TXY TYA, TYB, TYX, WIT, XAB	CLC

#### 2. Immediate

Instruction	Coding format
ADC, AND, CMP, EOR, LDA ORA, SBC	ADC    Acc,#imm
DIV, DIVS, MPY, MPYS, RMPA	DIV    #imm
CPX, CPY, LDX, LDY	CPX    #imm
CLP, SEP	CLP    #d8 CLP    C,I,M
LDT	LDT    #d8

## APPENDIX E. INSTRUCTION SETS BY ADDRESSING MODE

---

### 3. Accumulator

Instruction	Coding format
ASL, ASR, DEC, EXTS, EXTZ INC, LSR, ROL, ROR	ASL     Acc
RLA	RLA     #d8

### 4. Direct Bit

Instruction	Coding format
CLB, SEB	CLB     #imm,zz CLB     #imm,dp:zz

### 5. Absolute Bit

Instruction	Coding format
CLB, SEB	CLB     #imm,hhll CLB     #imm,dt:hhll

### 6. Stack

Instruction	Coding format
PHA, PHB, PHD, PHG, PHP PHT, PHX, PHY, PLA, PLB PLD, PLP, PLT, PLX, PLY	PHA
PEI	PEI     #d8
PSH, PUL	PSH     #d8 PSH     A,X,Y
PEA, PER	PEA     #d16

### 7. Relative

Instruction	Coding format
BCC, BCS, BEQ, BMI, BNE BPL, BVC, BVS	BCC     rr
BRA	BRA     rr
BRAL	BRAL    rrll

### 8. Block Transfer

Instruction	Coding format
MVN, MYP	MVN     d8,da

### 9. Direct

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC    Acc,zz
ORA, SBC, STA	ADC    Acc,dp:zz
DIV, DIVS, MPY, MPYS	DIV    ZZ
	DIV    dp:zz
ASL, CPX, CPY, DEC, INC LDX, LDY, LSR, ROL, ROR STX, STY	ASL    ZZ
	ASL    dp:zz
LDM	LDM    #imm,zz
	LDM    #imm,dp:zz

### 10. Direct Index X

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC    Acc,zz,X
ORA, SBC, STA	ADC    Acc,dp:zz,X
DIV, DIVS, MPY, MPYS	DIV    zz,X
	DIV    dp:zz,X
ASL, DEC, INC, LDY, LSR ROL, ROR, STY	ASL    zz,X
	ASL    dp:zz,X
LDM	LDM    #imm,zz,X
	LDM    #imm,dp:zz,X

### 11. Direct Index Y

Instruction	Coding format
LDX, STX	LDX    zz,Y
	LDX    dp:zz,Y

### 12. Direct Indirect

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC    Acc,(zz)
ORA, SBC, STA	ADC    Acc,(dp:zz)
DIV, DIVS, MPY, MPYS	DIV    (zz)
	DIV    (dp:zz)

## APPENDIX E. INSTRUCTION SETS BY ADDRESSING MODE

---

### 13. Direct Indirect Index X

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC Acc,(zz,X)
ORA, SBC, STA	ADC Acc,(dp:zz,X)
DIV, DIVS, MPY, MPYS	DIV (zz,X)
	DIV (dp:zz,X)

### 14. Direct Indirect Index Y

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC Acc,(zz),Y
ORA, SBC, STA	ADC Acc,(dp:zz),Y
DIV, DIVS, MPY, MPYS	DIV (zz),Y
	DIV (dp:zz),Y

### 15. Direct Indirect Long

Instruction	Coding format
ADCL, ANDL, CMPL, EORL	ADCL Acc,(zz)
LDAL, ORAL, SBCL, STAL	ADCL Acc,(dp:zz)
DIVL, DIVSL, MPYL, MPYSL	DIVL (zz)
	DIVL (dp:zz)

### 16. Direct Indirect Long Index Y

Instruction	Coding format
ADCL, ANDL, CMPL, EORL	ADCL Acc,(zz),Y
LDAL, ORAL, SBCL, STAL	ADCL Acc,(dp:zz),Y
DIVL, DIVSL, MPYL, MPYSL	DIVL (zz),Y
	DIVL (dp:zz),Y



### 17. Absolute

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC    Acc,hhll
ORA, SBC, STA	ADC    Acc,dt:hhll
DIV, DIVS, MPY, MPYS	DIV    hhll
	DIV    dt:hhll
ASL, CPX, CPY, DEC, INC LDX, LDY, LSR, ROL, ROR STX, STY	ASL    hhll
	ASL    dt:hhll
JMP, JSR	JSR    hhll
LDM	LDM    #imm,hhll
	LDM    #imm,dt:hhll

### 18. Absolute indexed X

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC    Acc,hhll,X
ORA, SBC, STA	ADC    Acc,dt:hhll,X
DIV, DIVS, MPY, MPYS	DIV    hhll,X
	DIV    dt:hhll,X
ASL, DEC, INC, LDY, LSR ROL, ROR	ASL    hhll,X
	ASL    dt:hhll,X
LDM	LDM    #imm,hhll,X
	LDM    #imm,dt:hhll,X

### 19. Absolute indexed Y

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC    Acc,hhll,Y
ORA, SBC, STA	ADC    Acc,dt:hhll,Y
DIV, DIVS, MPY, MPYS	DIV    hhll,Y
	DIV    dt:hhll,Y
LDX	LDX    hhll,Y
	LDX    dt:hhll,Y

## APPENDIX E. INSTRUCTION SETS BY ADDRESSING MODE

---

### 20. Absolute long

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC Acc,hhmml
ORA, SBC, STA	ADC Acc,lg:hhmml
DIV, DIVS, MPY, MPYS	DIV hhmml DIV lg:hhmml
JMPL, JSRL	JMPL hhmml JMPL lg:hhmml

### 21. Absolute long indexed X

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC Acc,hhmml,X
ORA, SBC, STA	ADC Acc,lg:hhmml,x
DIV, DIVS, MPY, MPYS	DIV hhmml,X DIV lg:hhmml,x

### 22. Stack pointer relative

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC Acc,d8,S
ORA, SBC, STA	
DIV, DIVS, MPY, MPYS	DIV d8,S

### 23. Stack pointer relative indirect indexed X

Instruction	Coding format
ADC, AND, CMP, EOR, LDA	ADC Acc,(d8,S),Y
ORA, SBC, STA	
DIV, DIVS, MPY, MPYS	DIV (d8,S),Y

### 24. Absolute indirect

Instruction	Coding format
JMP	JMP (hhl)

## 25. Absolute indirect indexed X

<b>Instruction</b>	<b>Coding format</b>
JMP, JSR	JMP (hhl,X)

## 26. Absolute indirect long

<b>Instruction</b>	<b>Coding format</b>
JMPL	JMPL (hhl)

## 27. Direct bit relative

<b>Instruction</b>	<b>Coding format</b>
BBC, BBS	BBC #imm,zz,rr
	BBC #imm,dp:ZZ,rr

## 28. Absolute bit relative

<b>Instruction</b>	<b>Coding format</b>
BBC, BBS	BBC #imm,hhl,rr
	BBC #imm,d:ihhl,rr

## E.2 Addressing Mode Relationship Table

Table E.1 shows the addressing mode selected by the assembler according to the data specified in the operand and the error messages output by the assembler when an invalid data combination is specified. Refer to this table when programming. The following symbols are used in the tables:

- : Checks the scope of address.
- \*: The specified addressing mode is selected unconditionally.
- : Page boundary is checked.
- ▲: The maximum addressing mode allowed for the instruction is used.
- PE: “Error 27: Page error” is output.
- VO: “Error 22: Value is out of range” is output.
- \*1: The vertical column shows the addressing mode specified with the pseudo instruction “.DT” and “.DP”.
- \*2: The horizontal column shows the addressing mode specified with the operand of the instruction.

**Table E.1 Addressing Mode Table**

		*1			LOC		EXT						
		OFF	IMM	SYM	REL	ABS	DPEXT <sup>1</sup>	DPEXT <sup>2</sup>	DTEXT <sup>3</sup>	DTEXT <sup>4</sup>	EXT		
DP: specified	IMM	○	○	○	○	○	○	○	○	○	○	○	
	SYM	○	○	○	○	○	○	○	○	○	○	○	
	LOC	REL	VO	*	*	*	*	*	*	*	*	*	*
		ABS	VO	○	○	*	○	*	*	*	*	*	*
	EXT	DPEXT <sup>1</sup>	VO	*	*	*	*	*	*	*	*	*	*
		DPEXT <sup>2</sup>	VO	PE	PE	PE	PE	PE	□	PE	PE	PE	PE
		DTEXT <sup>3</sup>	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO
		DTEXT <sup>4</sup>	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO
EXT		VO	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO	

## E.2 Addressing Mode Relationship Table

DT: specified	IMM		○	○	○	○	○	○	○	○	○	○
	SYM		○	○	○	○	○	○	○	○	○	○
	LOC	REL	VO	*	*	*	*	*	*	*	*	*
	EXT	ABS	VO	○	○	*	○	*	*	*	*	*
		DPEXT <sup>1</sup>	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO
		DPEXT <sup>2</sup>	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO
		DTEXT <sup>3</sup>	VO	*	*	*	*	*	*	*	*	*
		DTEXT <sup>4</sup>	VO	PE	PE	PE	PE	PE	PE	PE	□	PE
EXT		VO	VO	VO	VO	VO	VO	VO	VO	VO	VO	
DT: DP: not specified	IMM		▲	○	○	▲	○	▲	▲	▲	▲	▲
	SYM		▲	○	○	▲	○	▲	▲	▲	▲	▲
	LOC	REL	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲
	EXT	ABS	▲	○	○	▲	○	▲	▲	▲	▲	▲
		DPEXT <sup>1</sup>	▲	*	*	*	*	*	*	*	*	*
		DPEXT <sup>2</sup>	▲	PE	PE	PE	PE	PE	□	PE	PE	PE
		DTEXT <sup>3</sup>	▲	*	*	*	*	*	*	*	*	*
		DTEXT <sup>4</sup>	▲	PE	PE	PE	PE	PE	PE	PE	PE	□
EXT		▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲

## E.3 Selection of Addressing Mode

Addressing mode selection in RASM77 when using ".DP OFF" and ".DT OFF" is subject to the following rules.

Description format	Processing by RASM77
DP:label	Error
DP:symbol	Selects direct addressing mode
DP:absolute value	Selects direct addressing mode
DT:label	Error
DT:symbol	Selects absolute addressing mode
DT:absolute value	Selects absolute addressing mode

---

PART 2

**STRUCTURED PREPROCESSOR  
FOR 7700 FAMILY**



**PRE77 OPERATION MANUAL**

---

# Table of Contents

## Chapter 1. Organization of PRE77 Operation Manual

## Chapter 2. Overview

2.1 Function .....	2
2.2 Files Created .....	2

## Chapter 3. Source Program Coding Method

3.1 Structure of Source Program .....	6
3.2 Line Formats .....	7
3.2.1 Instruction Line .....	7
3.2.2 Structured Preprocessor Instruction Line .....	7
3.2.3 Pseudo Instruction Line .....	8
3.2.4 Macro Instruction Line .....	8
3.2.5 Comment Line .....	8
3.3 Field Coding Method .....	8
3.3.1 Symbol/Label Field .....	8
3.3.2 Op-code/Pseudo Instruction Field .....	9
3.3.3 Operand Field .....	9
3.3.4 Comment Field .....	9
3.4 Operand Field Coding Method .....	10
3.4.1 Data Format .....	10

## Chapter 4. Structured Preprocessor Instructions

4.1 Function of Structured Instructions .....	12
4.2 Statement Types .....	12
4.3 Coding Rules .....	14
4.4 Operators in Structured Instruction .....	19
4.5 Structured Instructions in Macros .....	20
4.6 RASM77 Instruction Lines, Pseudo Instruction Lines .....	20

## Chapter 5. Pseudo Instructions

5.1 Function of Pseudo Instructions .....	21
5.2 Preprocess Control .....	21

---

## Chapter 6. Operation

<b>6.1 Starting PRE77 .....</b>	<b>23</b>
<b>6.2 Input Parameters .....</b>	<b>23</b>
6.2.1 Source Filename .....	23
6.2.2 Command Parameters .....	23
<b>6.3 Input Method .....</b>	<b>25</b>
<b>6.4 Errors .....</b>	<b>26</b>
6.4.1 Error Types .....	26
6.4.2 Error Information .....	26
<b>6.5 Return Values to MS-DOS .....</b>	<b>27</b>
<b>6.6 Environment Variables .....</b>	<b>27</b>

## Appendix A. Error Messages

<b>A.1 System Error Messages .....</b>	<b>28</b>
<b>A.2 Preprocessor Error Messages .....</b>	<b>29</b>
<b>A.3 Warning Messages .....</b>	<b>32</b>

## Appendix B. Structured Preprocessor Instructions

<b>B.1 Conventions .....</b>	<b>33</b>
<b>B.2 Structured Preprocessor Instructions .....</b>	<b>33</b>
<b>B.3 Structured Preprocessor Instruction Syntax Diagram .....</b>	<b>56</b>

## Appendix C. Pseudo Instructions

<b>C.1 Conventions .....</b>	<b>69</b>
<b>C.2 Pseudo Instructions .....</b>	<b>69</b>



---

## List of Figures

Figure 2.1 Assembly File Example .....	3
Figure 2.2 Assembly File Example (Source Level Debug InformationOutput) .....	4
Figure 2.3 Tag File Example .....	5
Figure 6.1 Example of PRE77 Startup Command Line .....	25
Figure 6.2 MS-DOS Version Help Messages .....	25
Figure 6.3 Error Display Example .....	26

---

## List of Tables

Table 3.1 List of Operators .....	11
Table 4.1 Accumulator Bit Reference Reserved Words .....	14
Table 4.2 List of Generated Labels .....	17
Table 4.3 List of Generated Branch Instructions .....	17
Table 4.4 List of Operators that can be Used in Structured Instructions .....	19
Table 6.1 List of Command Parameters .....	24
Table 6.2 Listing of Error Levels .....	27
Table A.1 List of System Error Messages .....	28
Table A.2 List of Preprocessor Error Messages .....	29
Table A.3 List of Warning Messages .....	32

---

# CHAPTER 1

## Organization of PRE77 Operation Manual

The PRE77 Operation Manual consists of the following chapters:

- Chapter 2. Overview  
Describes the basic functions of the PRE77 and the files created by PRE77.
- Chapter 3. Source Program Coding Method  
Describes how to code source programs containing structured preprocessor codes.
- Chapter 4. Structured Preprocessor Instructions  
Describes the types of structure preprocessor instructions and how to code them.
- Chapter 5. Pseudo Instruction  
Describes the preprocessor instructions that are processed by PRE77.
- Chapter 6. Operation  
Describes how to enter PRE77 commands.
- Appendix A. Error Messages  
Lists error messages output by PRE77 along with explanation of the errors and actions to be taken.
- Appendix B. Structured Preprocessor Instructions  
Lists and explains all structured preprocessor instructions provided by PRE77.
- Appendix C. Pseudo Instructions  
Lists and explains all pseudo instructions provided by PRE77.

# CHAPTER 2

## Overview

The structured preprocessor allows the use of structured programming statements such as if and for which are not available in assembly language. This simplifies program development using the source level debugging functions of the 7700 Family debugger control software and improves development performance compared with development in straight assembly language.

### 2.1 Function

PRE77 converts programs written in 7700 Family structured preprocessor language into RASM77 assembly program. PRE77 can be used together with RASM77, LINK77<sup>1</sup>, and LIB77<sup>2</sup>. The following functions are provided:

1. Process RASM77 pseudo instructions “.INCLUDE”, “.DATA”, “.INDEX”, and “.EQU”
2. Process files included with “.INCLUDE”.
3. Generate assembly language source program file that can be assembled by RASM77.
4. Output “.CLINE”, “.FUNC”, “.ENDFUNC” that can be used for source line debugging with debugger.
5. Process structured preprocessor code within macro definitions.

In addition, the structure preprocessor generates a tag file<sup>3</sup> containing error descriptions (facilitates correction of preprocessor errors).

### 2.2 Files Created

PRE77 generates the following two types of files:

---

<sup>1</sup> LINK77 is the name of the 7700 Family linkage editor program.

<sup>2</sup> LIB77 is the name of the 7700 Family librarian program.

<sup>3</sup> This file is called a TAG file because it contains “tags” that show the location of errors and warnings.

1. Assembly language source program file for RASM77 (hereafter referred to as assembler file)
  - Contains RASM77 source level debugging pseudo instructions “.CLINE”, “.FUNC”, “.ENDFUNC”.
  - Generates Intel HEX format machine language data processed by RASM77 and LINK77.
  - The file extension is .A77.
  - Figures 2.1 and 2.2 show examples of assembly files output by PRE77.

```
; *** 7700 Family PREPROCESSOR V.5.00.00 ***
      .language      PRE77_Rev01
;      sample list
;
;FLAG_0 .EQU  0,000H
FLAG_0  .define  000H
;FLAG_1 .EQU  1,000H
FLAG_1  .define  000H
;FLAG_2 .EQU  2,000H
FLAG_2  .define  000H
;
      .EXT  WORK1
      .EXT  WORK2
;
      .SECTION  PROGRAM
;
;for [ DP:FLAG_0 ] == 1
..F1:
    BBC    #00001H,DP:FLAG_0,..F2
;    if [ DP:FLAG_1 ] == 1
    BBC    #00002H,DP:FLAG_1,..I3
;    [ DP:WORK1 ] = 0
    LDM    #0,DP:WORK1
;    if [ DP:FLAG_2 ] == 1
    BBC    #00004H,DP:FLAG_2,..I5
;    [ DP:WORK2 ] = 0
    LDM    #0,DP:WORK2
;    endif
..I5:
;    endif
..I3:
    BRA    ..F1
;next
..F2:
      .END
```

Figure 2.1 Assembly File Example

```
; *** 7700 Family PREPROCESSOR V.5.00.00 ***
    .language      PRE77_Rev01
    .source        sample.p77
;    sample list
;
;FLAG_0  .EQU    0,000H
FLAG_0   .define  000H
;FLAG_1  .EQU    1,000H
FLAG_1   .define  000H
;FLAG_2  .EQU    2,000H
FLAG_2   .define  000H
;
        .EXT     WORK1
        .EXT     WORK2
;
        .SECTION  PROGRAM
        .func     _sample_0
;
;for [ DP:FLAG_0 ] == 1
    .cline      12
..F1:
    BBC        #00001H,DP:FLAG_0,..F2
;    if [ DP:FLAG_1 ] == 1
    .cline      13
    BBC        #00002H,DP:FLAG_1,..I3
;        [ DP:WORK1 ] = 0
    .cline      14
    LDM        #0,DP:WORK1
;    if [ DP:FLAG_2 ] == 1
    .cline      15
    BBC        #00004H,DP:FLAG_2,..I5
;        [ DP:WORK2 ] = 0
    .cline      16
    LDM        #0,DP:WORK2
;    endif
    .cline      17
..I5:
;    endif
    .cline      18
..I3:
    BRA        ..F1
;next
    .cline      19
..F2:
        .endfunc     _sample_0
        .END
```

Figure 2.2 Assembly File Example (Source Level Debug Information Output)

### 2. Tag file

- This file contains error messages and warning messages generated during PRE77 processing.
- The file extension is .PTG.
- The tag file should be referenced when correcting errors with an editor.
- The tag file is output when the command parameter “-E” is specified.
- Figure 2.3 shows an example of a tag file.

```
SAMPLE.P77 120 ( TOTAL LINE 120 ) Error 9: else not associated with if
SAMPLE.P77 135 ( TOTAL LINE 135 ) Error 13: break not inside for, do or switch
SAMPLE.P77 140 ( TOTAL LINE 140 ) Error 6: Not in conditional block
```

**Figure 2.3 Tag File Example**

# CHAPTER 3

## Source Program Coding Method

### 3.1 Structure of Source Program

A PRE77 source program is made up of lines. Each source program line must comply with the following rules:

1. Each line must be complete by itself, and an instruction cannot be coded on more than one line.
2. Each line may contain no more than 256 characters. The PRE77 ignores coding beyond 256 characters.
3. Each line consists of the following fields:
  - Symbol/label field  
Label for referencing this line from other locations or symbol whose value is to be set by the .EQU pseudo instruction is coded in this field.
  - Op-code/pseudo instruction field  
7700 Family instruction mnemonic (hereafter referred to as op-code) or pseudo instruction is coded in this field.
  - Operand field  
Object of processing by op-code or pseudo instruction is coded in this field.
  - Comment field  
Specification in this field is not processed by PRE77 and the user can use this field for any purpose.

There are five types of lines.

1. Instruction line  
An instruction line specifies an 7700 Family instruction. This line is not processed by PRE77.
2. Structured preprocessor instruction line  
This line contains the 7700 Family structured preprocessor instruction. Assembly instruction lines are generated from this line.
3. Pseudo instruction line  
This line contains only pseudo instruction that is processed by PRE77.
4. Macro instruction line  
A macro instruction line specifies the macro definition. This line is processed by the assembler.



### 5. Comment line

A comment line is not processed by PRE77. Therefore, it can be used by the user for any purpose.

## 3.2 Line Formats

This section describes the format of each type of line. The following conventions are used for these descriptions:

1.  $\triangle$  and  $\blacktriangle$  specify space or tab code.  $\triangle$  is required, and  $\blacktriangle$  is optional.
2. A colon (:) is required when coding a label.

### 3.2.1 Instruction Line

Shown below is the format of an instruction line:

$\blacktriangle$  Label:  $\blacktriangle$  Op-code  $\triangle$  Operand  $\blacktriangle$  ; Comment <RET>

$\blacktriangle$ † Op-code  $\triangle$  Operand  $\blacktriangle$  ; Comment <RET>

† Because PRE77 identifies each instruction as reserved word, a line can begin with an op-code if there is no label.

### 3.2.2 Structured Preprocessor Instruction Line

Shown below is the format of the structured preprocessor instruction line.

$\blacktriangle$  Label:  $\blacktriangle$  structured-preprocessor-instruction  $\triangle$ condition-expression $\blacktriangle$  ; Comment <RET>

$\blacktriangle$ †structured-preprocessor-instruction  $\triangle$ condition $\blacktriangle$  ; Comment <RET>

1. Label field  
Code the label for referencing this line from other parts of the program.  
PRE77 processes the structured preprocessor instruction and outputs the content of this field unchanged when generating the 7700 Family assembly language code.
2. Structured preprocessor instruction field  
Code the 7700 Family structure preprocessor instruction. The structure preprocessor makes no distinction between uppercase and lowercase characters. Therefore, IF and if are both valid.
3. Condition expression field  
Code the condition to be processed by the structured preprocess.

### 4. Comment field

This field is not processed by the structure preprocessor. The user is free to use this field.

Note:

† Because PRE77 identifies each instruction as reserved word, a line can begin with an op-code if there is no label.

### 3.2.3 Pseudo Instruction Line

Shown below is the format of pseudo instruction line:

▲Label: ▲ Pseudo-op △ Operand ▲ ; Comment <RET>

▲ Symbol △ .EQU △ Operand ▲ ; Comment <RET>

▲ ‡ Pseudo-op △ Operand ▲ ; Comment <RET>

Note:

‡ Because RASM77 identifies each pseudo instruction as reserved word, a line can begin with an op-code if there is no label.

### 3.2.4 Macro Instruction Line

The format of a macro instruction line is shown below. This line is not processed by PRE77 (Refer to Chapter 6 and Appendix E for details concerning this line).

▲ Macro name:▲ Macro Instruction △ Operand ▲ ; Comment <RET>

### 3.2.5 Comment Line

Comment line must begin with a semicolon (;). Shown below is the format of a comment line:

▲ ; Comment <RET>

## 3.3 Field Coding Method

### 3.3.1 Symbol/Label Field

PRE77 manages symbols and labels separately, but the same coding format applies to both. The coding format is described below.

1. A symbol or label can be specified using alphanumeric characters, special characters, underline (  ) and question mark (?). The first character must be an alphabetic or special character.

2. Reserved words cannot be used as names. PRE77 processes register names, flag names, op-code, pseudo instruction and operand description instructions (including DP, DT, LG) as reserved words.
3. Uppercase and lowercase are distinguished. Therefore, "BIG" and "Big" are recognized as different names.
4. A label or symbol may be no more than 255 characters long.
5. The following labels beginning with '..' (two periods) must not be used because they are labels generated by PRE77. Other labels beginning with '..' must also be avoided because they may be used by PRE77 or RASM77 in the future.
  - ..D0 to ..D65535
  - ..F0 to ..F65535
  - ..I0 to ..I65535
  - ..S0 to ..S65535
6. When coding a label, it must be followed immediately by a colon (:). However, a warning is issued if a colon is coded immediately after a symbol.

### 3.3.2 Op-code/Pseudo Instruction Field

A 7700 Family instruction mnemonic or a pseudo instruction is specified in the op-code/pseudo instruction field. The specification format is described below.

1. No distinction is made between uppercase and lowercase characters for instruction mnemonic and pseudo instructions. Thus, both "NOP" and "nop" mean the same.

### 3.3.3 Operand Field

The target of instruction or pseudo instruction is specified in the operand field. The specification format is described below.

1. If there are two or more operand data, they must be delimited by comma (,).
2. Space or tab code may be specified on either side of a comma.

### 3.3.4 Comment Field

Any user information may be specified in the comment field. The specification format is described below.

1. A comment field must begin with a semicolon (;).
2. Any character may be used in the comment field.

# 3.4 Operand Field Coding Method

### 3.4.1 Data Format

Operand field may be specified with data in any of the following four data formats:

#### 1. Numeric constant

- A numeric constant can be specified as a positive or negative value by using the '+' or '-' operator as prefix. If neither '+' nor '-' is specified, the numeric constant is processed as a positive value.
- A binary, octal, decimal or hexadecimal number may be specified as a numeric constant.
- When specifying a binary numeric constant, the value must be followed by 'B' or 'b'.

Example:        `DATA .EQU 100110B`

- When specifying an octal numeric constant, the value must be followed by 'O' or 'o'.

Example:        `DATA .EQU 70o`

- When specifying a decimal numeric constant, only an integer value can be specified.

Example:        `DATA .EQU 100`

- When specifying a hexadecimal numeric constant, the value must be followed by 'H' or 'h'. If the hexadecimal value begins with an alphabetic character (A to F), 0 must be specified at the beginning.

Example 1:        `.EQU 64H`

Example 2:        `.EQU 0ABH`

#### 2. Character string constant

- Any ASCII code character may be used in a character string constant.
- Character string constant must be enclosed between single quotes (' ') or double quotes (" ").

Example:        `CHAR .EQU 'A' Sets 41H.`

#### 3. Label or symbol

- A label normally indicates an address, but they do not have any value because PRE77 does not process labels. A symbol has a 32 bit data value.

#### 4. Expression

- Numeric expression can be specified as a combination of operators, numeric constants, character string constants, labels or symbols.
- An expression is calculated from left to right. (No operator priorities are recognized.)

Example 1:        `2*3 => Result is 6.`

Example 2:        `2+6/2 => Result is 4.`

Table 3.1 lists the operators that may be used with PRE77.

Table 3.1 List of Operators

Operator <sup>1</sup>	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder of division
<<	Left shift
>>	Right shift
&	Logical AND on bits
	Logical OR on bits
^	Logical exclusive-OR on bits
+	Unary operator specifying a positive number
-	Unary operator specifying a negative number
~	Unary operator specifying bit inversion
BANK <sup>3</sup>	Unary operator to extract high-order 8 bits of label or symbol
OFFSET <sup>3</sup>	Unary operator to extract low-order 16 bits of label or symbol

Note:

1. Operation is executed from left to right. (No operator priorities are recognized.)  
 Example 1:  $2+6/2 \Rightarrow$  Result is 4.  
 Example 2:  $2*3 \Rightarrow$  Result is 6.
2. Multiple unary operators written in one line are not accepted.

# CHAPTER 4

## Structured Preprocessor Instructions

### 4.1 Function of Structured Instructions

Structured instructions enable the use of structured programming statements such as if and for rather than goto type statements such as assembly language branch and jump instructions. With structured programming, there is no need to search for branch destination.

### 4.2 Statement Types

The structured preprocessor language consists of the following seven types of statements. Appending “!” or “!!” results in corresponding long branch. The details of each statement is described in Appendix B.

1. Assignment statement  
Assigns the right hand term to the left hand term.
2. if - (!(l))else - endif statement
3. lif - (!(l))else - endif statement
4. llif - (!(l))else - endif statement  
The if statement changes the flow of control into two separate directions determined by the condition expression.
5. for - next statement
6. lfor - next statement
7. llfor - next statement  
The for statement controls program loop and repeats a group of statements while the specified condition is true.
8. do - while statement
9. ldo - while statement
10. lldo - while statement  
The do statement repeats a group of statements while the specified condition is true.

11. switch - case - ends statement

12. lswitch - case - ends statement

13. llswitch - case - ends statement

The switch statement passes control to one of several statements according to the value of the condition expression.

14. break statement

15. lbreak statement

16. llbreak statement

The break statement stops execution of for, do, or switch construct and passes control to the next statement.

17. continue statement

18. lcontinue statement

19. llcontinue statement

The continue statement creates a dummy null statement at the end of the innermost for or do construct containing the continue statement and passes control to that statement.

20. goto statement

21. lgoto statement

22. llgoto statement

The goto statement causes an unconditional jump to any address that is explicitly indicated in the program.

## 4.3 Coding Rules

The rules for coding in structured preprocessor language are described below.

1. When coding memory that is referenced in each 7700 Family addressing mode in the condition expression of assignment statement or control statements (such as if), it must be enclosed in “[ ]” or “{ }”.

Example 1:

```
[ WORK ] = 10
```

Example 2:

```
if [ WORK ]
:
else
:
endif
```

2. When coding any bit that can be referenced by a bit symbol in the condition expression of assignment statement or control statements (such as if), it must be enclosed in “[ ]” or “{ }”. However, the following reserved words are provided to reference an accumulator bit. In this case, the “[ ]” or “{ }” are not necessary. Furthermore, no distinction is made between uppercase and lowercase characters. Therefore, BIT\_A0 and bit\_a0 are both valid. Table 4.1 shows the accumulator bit reference reserved words.

**Table 4.1 Accumulator Bit Reference Reserved Words**

BIT_A0	Accumulator A bit 0	BIT_B0	Accumulator B bit 0
BIT_A1	Accumulator A bit 1	BIT_B1	Accumulator B bit 1
BIT_A2	Accumulator A bit 2	BIT_B2	Accumulator B bit 2
BIT_A3	Accumulator A bit 3	BIT_B3	Accumulator B bit 3
BIT_A4	Accumulator A bit 4	BIT_B4	Accumulator B bit 4
BIT_A5	Accumulator A bit 5	BIT_B5	Accumulator B bit 5
BIT_A6	Accumulator A bit 6	BIT_B6	Accumulator B bit 6
BIT_A7	Accumulator A bit 7	BIT_B7	Accumulator B bit 7
BIT_A8	Accumulator A bit 8	BIT_B8	Accumulator B bit 8
BIT_A9	Accumulator A bit 9	BIT_B9	Accumulator B bit 9
BIT_A10	Accumulator A bit 10	BIT_B10	Accumulator B bit 10
BIT_A11	Accumulator A bit 11	BIT_B11	Accumulator B bit 11
BIT_A12	Accumulator A bit 12	BIT_B12	Accumulator B bit 12
BIT_A13	Accumulator A bit 13	BIT_B13	Accumulator B bit 13
BIT_A14	Accumulator A bit 14	BIT_B14	Accumulator B bit 14
BIT_A15	Accumulator A bit 15	BIT_B15	Accumulator B bit 15

A bit symbol is defined with the pseudo instruction “.EQU”. This line is output as a command line to the RASM77 assembler file. However, RASM77 coding lines are passed to RASM77 to be processed. PRE77 processes only pseudo instruction “.EQU” that is coded in the following format.



Example 1: In the case of memory bit

```

BITSYM    .EQU 1,1000h    ; Defines a bit symbol
    if [ DP:BITSYM ]
        :
    else
        :
    endif

```

Example 2: In the case of accumulator bit

```

    if BIT_A0
        :
    else
        :
    endif
    if BIT_B3
        :
    else
        :
    endif

```

3. 7700 Family registers are coded as follows in the condition expression of assignment statement or each control statement (such as if statement). No distinction is made between uppercase and lowercase characters so that A and a are both valid.

A	Accumulator A	B	Accumulator B
X	Index register X	Y	Index register Y
S	Stack pointer	DPR	Direct page register
DT	Data bank register	PS	Processor status register

4. The flags in 7700 Family status registers are coded as follows in the condition expression of assignment statement or each control statement (such as if statement). No distinction is made between uppercase and lowercase characters so that C and c are both valid.

C	Carry flag	Z	Zero flag
I	Interrupt disable flag	D	Decimal operation mode flag
XF	Index register length selection flag	M	Data length selection flag
V	Overflow flag	N	Negative flag
IPL	Processor interrupt level		

Example 1:

```
C = 1
```

Example 2:

```
if C == 0
:
else
:
endif
```

5. PRE77 treats the following coding (forward reference) as label (expand into LDA instruction). Therefore, if "BITSYM" is later defined as bit symbol, the expanded code will not match (this results in PRE77 error). In this case, rewrite the program to define the bit symbol (BITSYM) before it is referenced.

Example:

```
if [ BITSYM ]
:
else
:
endif
:
BITSYM .EQU 1,10h
```

6. In the following code, PRE77 passes the pseudo instruction ".DEFINE" to RASM77 without processing. Therefore, if a character string defined as character string appears in a structured preprocessor code, it is treated as a normal character string. In this case RASM77 outputs an error for "OTHER".

Example:

```
OTHER .DEFINE else
:
if BIT_A0
:
OTHER
:
endif
```

Refer to Appendix B for details concerning structure preprocessor instructions.

7. PRE77 generates the following labels when converting structured preprocessor codes to RASM77. The user must not use these labels.

An error will occur if the number of labels exceeds the allowed maximum (65535). Table 4.2 shows the labels generated by PRE77.

**Table 4.2 List of Generated Labels**

..D0 - ..D65535	labels for do - while
..F0 - ..F65535	labels for for - next
..I0 - ..I65535	labels for if - else - endif
..S0 - ..S65535	labels for switch - case - ends

8. Structured preprocessor instructions that generate branch instructions produce different branch instruction according to the instruction in order to use memory efficiently. Table 4.3 shows the generated branch instructions.

**Table 4.3 List of Generated Branch Instructions**

Control Statement	Generated Instruction	Instruction Length
if,for,do,switch,break,continue,goto	BRA	2
lif,lfor,lido,lswitch,lbreak,lcontinue,lgoto	BRAL	3
llif,llfor,lldo,llswitch,llbreak,llcontinue,llgoto	JMPL	4

9. Macro arguments cannot be coded as the operand of the following pseudo instructions or structured preprocessor instructions because PRE77 does not perform macro expansion.

- Operand of .INDEX
- Operand of .DATA
- Operand of .EQU
- Condition expression of structured command
- CASE constant

10. When converting a condition expression to assembly language, a code using register A may be generated. Note that the following code will result in code using register A.

- When condition expression is coded only with memory variable or memory bit variable.

Example:

```
if [MEM]
  jar subl
endif
```

- Memory bit variable assignment statement is coded in absolute long addressing mode.

## CHAPTER 4. STRUCTURED PREPROCESSOR INSTRUCTIONS

---

11. Assembly language instruction using register A will be generated if an expression containing X and Y registers is coded.
12. PRE77 does not process the macro instruction itself. Therefore, structured descriptions using macro arguments cannot be processed correctly.
13. The codes generated when writing multiply/divide instructions in the structured description language are processed using the data length that is set in the 7700 family's data length select flag. Consequently, the 8 or 16 high-order bits of the operation result are not taken into account. For example, when using the value of 8 or 16 high-order bits derived from a multiply operation, write an instruction to store the content of the B register in memory immediately after the line where the multiply instruction is written, as shown below.

```
                .data      16
mem1:           .blkw      2
mem2:           .blkw      1
mem3:           .blkw      1
                [mem1] = [mem2] * [mem3]
                [mem1+2] = B
```

## 4.4 Operators in Structured Instruction

Table 4.4 shows the operators that can be used in structured preprocessor instructions.

**Table 4.4 List of Operators that can be Used in Structured Instructions**

Type	Operator	Description
Unary Operator	+	Indicates a positive number
	-	Indicates a negative number
	~	Takes the complement of 1
	++	Increment
	--	Decrement
Diacic Operator	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
	%	Modulo division
	&	Logical AND on bits
		Logical OR on bits
	^	Logical exclusive OR on bits
	&&	Logical AND
		Logical OR
	<<	Shift left
	>>	Shift right
	Comparison Operator	>
<		Greater than
==		Equal
!=		Not equal
<=		Less than or equal
>=		Greater than or equal

Note:

1. All operations that generate the 7700 Family instructions are processed as unsigned numeric values, and the bit lengths are processed using the bit length that is specified by ".DATA" or ".INDEX" when the processing is performed. However, if the result of an operation (only + or -) is subjected to comparison, etc, whether the result has overflowed is not taken into account. Therefore, when performing a comparison of whether large or small as shown below, be careful not to cause an overflow. (In the example below, if the content of work is FE16, the operation results in 0016, so that conditions are not met.)

Example:

```
.DATA16
if [work] + 2 > 10
    :
else
    :
endif
```

2. Several if statements can be grouped using diadic operators "&&" or "||".

Example:

```
if [ WORK ]          if [ WORK ] && [ WORK2 ]
    if [ WORK2 ]      :
        :            :
    endif            :
endif                endif
```

## 4.5 Structured Instructions in Macros

Structured instructions are allowed within macros. In this case, the labels generated by the structure preprocessor within macro definition are defined as macro local labels with the ".LOCAL" instruction.

PRE77 also generates codes for structured instructions within user macro definitions, but does not perform macro expansion.

## 4.6 RASM77 Instruction Lines, Pseudo Instruction Lines

PRE77 outputs RASM77 instruction lines and pseudo instruction lines to the RASM77 assembly source file without any change. Therefore, these lines are not checked for errors.

# CHAPTER 5

## Pseudo Instructions

### 5.1 Function of Pseudo Instructions

Pseudo instructions instruct the preprocessor or assembler to generate the target machine instruction data<sup>1</sup>.

The pseudo instructions that can be coded in source program using the structure preprocessor language are the same as those allowed for RASM77. However, PRE77 processes only the following pseudo instructions. The functions of pseudo instructions processed by PRE77 are not necessary the same as that for RASM77.

#### 1. Preprocessor control

```
.CLINE .DATA .END .ENDFUNC .EQU
.FUNC .INCLUDE .INDEX .SECTION .SOURCE
```

### 5.2 Preprocess Control

PRE77 processes pseudo instructions to control the preprocessing operation. Therefore, the functions of the pseudo instructions are not necessary the same as that for RASM77.

**.CLINE**

Defines line numbers necessary for source level debugging.

**.DATA**

Declares the default value of the data length selection flag (m).

**.END**

Declares the end of preprocess. PRE77 does not process lines following this line.

---

<sup>1</sup> Pseudo instructions that specify the default value to assembler are referred to as “declaration” and pseudo instructions that affect the output file are referred to as “specification”.

### **.EQU**

Sets a numeric value (double word) to a symbol.

### **.FUNC .ENDFUNC**

Declares the start and end of function necessary for source level debugging. These pseudo instruction are generated by PRE77.

### **.INCLUDE**

Loads another file in place of this pseudo instruction. This pseudo instruction is used to divide large source program into manageable sections. PRE77 allows this pseudo instruction to be nested up to 9 levels.

### **.INDEX**

Declares the default value of the index register length flag (x).

### **.SECTION**

Specifies the name of the program section following this line. This pseudo instruction must be used to specify the section name before starting a program.

### **.SOURCE**

Declares the source file name necessary for source level debugging. This pseudo instruction is generated by PRE77.



---

# CHAPTER 6

## Operation

### 6.1 Starting PRE77

Before PRE77 can be executed, the following information (input parameters) must be input:

1. Structured preprocessor source program filename (required)
2. Command parameters

### 6.2 Input Parameters

#### 6.2.1 Source Filename

1. The name of the source file to be processed by the structured preprocessor is specified. Source filename must always be specified. Only one source filename may be specified.
2. If the file extension (.P77) is omitted, .P77 is selected as default.
3. By specifying a full filename, files with other file extensions can be processed by PRE77. However, an error will occur if '.A77' is specified as the file extension because PRE77 uses '.A77' as the extension of the generated files.
4. Filename can be specified with directory path. If only filename is specified, PRE77 processes a file in the current drive's current directory. The following example shows an example of assembling TEST.P77 in directory WORK on drive C.

Example: A>PRE77 C:\WORK\TEST<RET>

#### 6.2.2 Command Parameters

1. Command parameters may be specified in either uppercase or lowercase.
2. The same command parameter may be specified more than once. Each parameter must be delimited by a space.

Table 6.1 summarizes the functions of command parameters.

## CHAPTER 6. OPERATION

---

**Table 6.1 List of Command Parameters**

Command parameter	Description
-.	Suppresses output of all messages to the screen.
-C	Performs processing to enable source level debugging of structured preprocessor instructions.
-E	Creates a TAG file when an error occurs. With the PC version, the program specified immediately after this option is started using the tag file as argument. With the UNIX version, an error occurs if anything is specified immediately after this option.
-L	When the structured instruction is converted into the assembly instructions, labels are generated for the operand of the branch instruction. If it is no specification, a relative address value is generated for the operand of the branch instruction.
-O	Specifies the output destination path for the file to be created. The full path must be specified.
-RASM77	Starts RASM77 after the preprocessor terminates. The string specified after this command parameter is passed to RASM77 as command parameters. The following example starts RASM77 after processing "SAMPLE.P77" and obtains the print file output by RASM77.  <pre>A&gt;PRE77 SAMPLE -RASM77 -L</pre> The -C parameter is automatically specified when RASM77 is started by PRE77.
-S	The comment which shows to converted the structured instruction into the assembly language are output.
-V	The version No. of PRE77 is output to the screen and the command is terminated.

Note:

1. If the "-O" option is not specified, PRE77 generates the assembly file in the directory where the structured description source exists.

## 6.3 Input Method

PRE77 is started by entering a command line after the MS-DOS prompt. Figure 6.1 illustrates entry of PRE77 startup command. If there is an error in the command line, a help screen shown in Figure 6.2 is output and the preprocessor is canceled.

```
A>PRE77 TESTNAME -C <RET>
      ↑      ↑
Name of source  Command parameters
to be reprocessed
```

**Figure 6.1 Example of PRE77 Startup Command Line**

```
7700 Family PREPROCESSOR V.5.00.00
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Usage: pre77 <filename> [-.][-e][-o][-rasm77 <rasm77 options>]

-.      : all messages suppressed
-c      : source line information output to .A77 file
-e      : make tag file and start editor ( syntax -e or -eEDITOR_NAME )
-o      : select drive and directory for output ( syntax -oA:\WORK )
-l      : make structured label
-s      : structured block message output to .a77 file
-v      : pre77 version display
-rasm77 : rasm77 option select
```

**Figure 6.2 PC Version Help Messages**

# 6.4 Errors

## 6.4.1 Error Types

The following types of errors may occur during execution of PRE77:

1. OS errors  
Errors related to the environment in which PRE77 is executed. These errors include disk and memory shortages.
2. PRE77 command line input errors  
These are the errors in PRE77 startup command line input.
3. Preprocessor source file contents errors  
These are errors in the contents of the source file being preprocessed (including files included with the pseudo instruction `.INCLUDE`).

When PRE77 detects an error, it outputs error information in the format shown in Figure 6.3.

```
7700 Family PREPROCESSOR V.5.00.00
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.
now processing ( SAMPLE.P77 ) ← Name of file to be processed.
-----*-----*--          ← Indicates the processing progress with '-' for every 100 lines
                                and '*' for every 500 lines.)

if
SAMPLE.P77 1240 ( TOTAL LINE 1240 ) Error 8: Questionable syntax
```

**Figure 6.3 Error Display Example**

## 6.4.2 Error Information

Information about the errors and warnings generated when executing PRE77 is output to the assembly source file that is generated by PRE77.

## 6.5 Return Values to OS

PRE77 returns the error level shown in Table 6.2 to indicate the execution result to OS.

**Table 6.2 Listing of Error Levels**

Error level	Execution result
0	Normal termination
1	Preprocessor source file contents error
2	PRE77 command input error
3	OS error
4	Force termination by ^C (control C)

## 6.6 Environment Variables

PRE77 uses the following MS-DOS environment variables:

1. TMP77

This variable specifies the name of the directory in which temporary files are created during preprocessing. If this environment variable is not set, the temporary files are created in the current directory.

2. INC77

This variable specifies the directory of the files included during preprocessing. If the file specified with the .INCLUDE pseudo instruction cannot be found, it is loaded from this directory. However, this environment variable is ignored if the ".INCLUDE" operand specifies a path.

3. BIN77

This variable specifies the directory in which RASM77 is searched when RASM77 is started from the preprocessor. If RASM77 is not in the current directory, the directory specified with this environment variable is searched.

# APPENDIX A

## Error Messages

### A.1 System Error Messages

When a system error is detected during preprocessing, PRE77 outputs an error message on the screen and cancels processing. Table A.1 lists the system error messages.

**Table A.1 List of System Error Messages**

<b>Error Message</b>	<b>Description and User Action</b>
Usage:pre77.....	Command input is invalid. ⇒ Check the HELP screen, and reenter the command.
Can't open xxx	File cannot be found. ⇒ Check the source filename, and reenter correctly.
Can't create xxx	File cannot be created. ⇒ Check the -o parameter specification, and reenter correctly.
Out of disk space	Disk space is insufficient for file output. ⇒ Provide sufficient free space on the disk.
Can't find rasm77	RASM77.EXE cannot be found. ⇒ Copy RASM77.EXE to the current directory or a directory specified by MS-DOS command path.
Can't find command.com for execute xxx	COMMAND.COM file necessary to start the editor specified by the -E option cannot be found. ⇒ Check MS-DOS command path specification.
Out of heap space	Memory space is insufficient to execute PRE77. ⇒ Reduce the number of symbols or labels.

## A.2 Preprocessor Error Messages

When a preprocessor error is detected, an error message is output on the screen. Table A.2 lists the preprocessor error messages.

**Table A.2 List of Preprocessor Error Messages**

<b>Error no.</b>	<b>Error message</b>	<b>Description</b>
1	Division by 0	A division by 0 is performed. ⇒ Check the coding of expressions.
2	Nesting error	Nesting level exceeds the allowed maximum. ⇒ Change the program and reduce the number of nests to within the allowed levels.
3	No ';' at the top of comment	There is no ';' in the comment field. ⇒ Code a ';' at the beginning of the comment field.
4	Operand is expected	A required operand is missing. ⇒ Check the operand coding.
5	Questionable syntax	There is an error in the mnemonic coding. ⇒ Check the mnemonic coding.
6	Label or symbol is reserved word	A reserved word is used as label or symbol. ⇒ Change the label or symbol.
7	Value is out of range	The value exceeds the allowed limit. ⇒ Check the operand coding.

## APPENDIX A. ERROR MESSAGES

Error no.	Error message	Description
8	ELSE not associated with IF	There is no IF statement corresponding to an ELSE statement. ⇒ Check the program.
9	ENDIF not associated with IF	There is no IF statement corresponding to an ENDIF statement. ⇒ Check the program.
10	NEXT not associated with FOR	There is no FOR statement corresponding to a NEXT statement. ⇒ Check the program.
11	WHILE not associated with DO	There is no DO statement corresponding to a WHILE statement. ⇒ Check the program.
12	ENDS not associated with SWITCH	There is no ENDS statement corresponding to a SWITCH statement. ⇒ Check the program.
13	BREAK not inside FOR, DO or SWITCH	The location of the BREAK statement is invalid. ⇒ Check the program.
14	CONTINUE not inside FOR or DO	The location of the CONTINUE statement is invalid. ⇒ Check the program.
15	CASE not inside SWITCH	The CASE statement is outside the scope of SWITCH statement. ⇒ Check the program.
16	DEFAULT not inside SWITCH	The DEFAULT statement is outside the scope of SWITCH statement. ⇒ Check the program.
17	Duplicate CASE value	The same value is duplicated as CASE value. ⇒ Check the program.
18	More than one DEFAULT	There is more than one DEFAULT in a SWITCH construct. ⇒ Check the program.
19	Bit length is different type	The data cannot be processed with the bit length specified with .INDEX or .DATA. ⇒ Change the bit length.



## A.2 Preprocessor Error Messages

Error no.	Error message	Description
20	Label or symbol is multiple defined	The same symbol is defined more than once with the pseudo instruction .EQU. ⇒ Check the program.
21	No .END statement	There is no pseudo instruction .END in the source file. ⇒ Code .END in the source file.
22	Illegal assign	An invalid assignment statement. ⇒ Check the program.
23	No endif statement	There is no endif statement corresponding to an if statement. ⇒ Check the program.
24	No next statement	There is no next statement corresponding to a for statement. ⇒ Check the program.
25	No while statement	There is no while statement corresponding to a do statement. ⇒ Check the program.
26	No ends statement	There is no ends statement corresponding to a switch statement. ⇒ Check the program.
27	can't include .a77 file	The extension of the specified include file is .a77. ⇒ Specify extension other than .a77 for an include file.
28	.ENDM not associated with macro	The ".ENDM" for the macro instruction is not written. ⇒ Write ".ENDM."
29	No .SECTION statement	The ".SECTION" instruction is preceded by a statement that generates instruction code. ⇒ Write ".SECTION" before a statement to generate instruction code.

## A.3 Warning Messages

When a warning condition is detected, a warning message is output to the screen. Table A.3 shows a list of possible warning messages.

**Table A.3 List of Warning Messages**

<b>Warning no.</b>	<b>Warning message</b>	<b>Description</b>
1	Statement has not effect	The statement is meaningless.
2	Not CASE values for SWITCH statement	There is no CASE statement within a SWITCH construct.
3	Statement not preceded by CASE or DEFAULT	There is an instruction before a CASE or DEFAULT statement in a SWITCH construct.
4	.EQU operand is not symbol	The operand of an .EQU instruction is not a symbol.
5	.END statement in include file	The pseudo instruction .END is coded in an include file.
6	Different DATA or INDEX length	The bit length specified with .DATA or .INDEX does not match the actual length.

# APPENDIX B

## Structured Preprocessor Instructions

### B.1 Conventions

The structured instructions available with RASM77 are listed below in alphabetical order. The following conventions are used in describing each pseudo instruction:

1. Space or tab code is indicated by  $\triangle$  or  $\blacktriangle$ .  $\triangle$  is a required space or tab code, and  $\blacktriangle$  is an optional space or tab code.
2. A colon (:) is required when specifying a label.

### B.2 Structured Preprocessor Instructions

### BREAK

BREAK statement

---

**Format:**

▲[label:]▲BREAK

**Description:**

- The break statement stops execution of the corresponding for, do or switch statement and passes control to the next statement.
- It can be use only within a for, do, or switch construct.
- A relative address branch instruction BRA is generated.

**Example:**

```
for [ flag1 ]
    if [ flag2 ]
        break
    endif
    jsr  output
next
```

---

**CONTINUE****CONTINUE statement**

---

**Format:**

▲[label:]▲CONTINUE

**Description:**

- The continue statement creates a dummy null statement at the end of the innermost for or do construct containing this statement and passes control to that statement.
- It can be use only within a for or do construct.
- A relative address branch instruction BRA is generated.

**Example:**

```
for [ flag1 ]
  if [ flag2 ]
    continue
  endif
  jsr  output
next
```

### DO - WHILE

DO statement

---

#### Format:

```
▲[label:]▲DO  
    <statement>  
▲[label:]▲WHILE△condition-expression
```

#### Description:

- The do statement repeats a group of statements while the specified condition-expression is true. The decision to repeat or not is made after executing the statement. Therefore, the do statement is useful in cases where the repeated statements are to be executed once more after the condition is satisfied.
- An endless loop is formed if “ever” is specified as the condition expression.
- Refer to the syntax diagram for the details concerning the condition expression.

#### Example:

```
do  
    jsr    output  
while [ flag ]
```

**Format:**

▲[label:]▲FOR△condition-expression  
    <statement>  
▲[label:]▲NEXT

**Description:**

- The for statement repeats a group of statements while the specified condition expression is true.
- An endless loop is formed if “ever” is specified as the condition expression.
- Refer to the syntax diagram for the details concerning the condition expression.

**Example:**

```
for [ flag ]  
    jsr  output  
next
```

### GOTO

GOTO statement

---

**Format:**

▲[label:]▲GOTO

**Description:**

- The goto statement causes an unconditional jump to any address that is explicitly indicated in the program.
- This statement can be written at any position in the program.
- Use a label to specify the jump address.
- Relative address branch instruction BRA is generated.

**Example:**

```
    for  [flag1]
      if  [flag2]
        goto    LAB1
      endif
      jsr  output
LAB1:
      jsr  inout
    next
```



---

**IF - (ELSE) - ENDIF**IF statement

---

**Format:**

```
▲[label:]▲IF△condition-expression
    <statement>
▲[[label:]▲ELSE]
    <statement>
▲[label:]▲ENDIF
```

**Description:**

- The if statement changes the flow of control into two directions. The branch direction is determined by the condition expression. The branch is made according to whether the condition expression results in zero (false) or non-zero (true). If true, the immediately following instruction is executed, if false, the instruction following else is executed if else is coded or instruction following endif is executed if else is not coded.
- The else part may be omitted.
- There is no limit to the if statement nesting level.
- When if statements are nested, the closest if and else statements are paired.
- Refer to the syntax diagram for the details concerning the condition expression.

**Example:**

```
if [ flag ]
    [ work ] = 1
else
    [ work ] = 2
endif
```

### LBREAK

LBREAK statement

---

**Format:**

▲[label:]▲LBREAK

**Description:**

- The lbreak statement stops execution of the corresponding for, do or switch construct and passes control to the next statement.
- It can be use only within a for, do, or switch construct.
- A relative address branch instruction BRAL is generated.

**Example:**

```
for [ flag1 ]
    if [ flag2 ]
        lbreak
    endif
    jsr  output
next
```

### LCONTINUE

LCONTINUE statement

---

**Format:**

▲[label:]▲LCONTINUE

**Description:**

- The lcontinue statement creates a dummy null statement at the end of the innermost for or do construct containing this statement and passes control to that statement.
- It can be use only within a for or do construct.
- A relative address branch instruction BRAL is generated.

**Example:**

```
for [ flag1 ]
  if [ flag2 ]
    lcontinue
  endif
  jsr  output
next
```

### LDO - WHILE

LDO statement

---

**Format:**

▲[label:]▲LDO

<statement>

▲[label:]▲WHILE△condition-expression

**Description:**

- The ldo statement repeats a group of statements while the specified condition-expression is true. The decision to repeat or not is made after executing the statement. Therefore, the ldo statement is useful in cases where the repeated statements are to be executed once more after the condition is satisfied.
- An endless loop is formed if “ever” is specified as the condition expression.
- Refer to the syntax diagram for the details concerning the condition expression.

**Note:**

- The generated branch instruction BRA is output as BRAL.

**Example:**

```
ldo
    jsr  output
while [ flag ]
```

**LFOR - NEXT**LFOR statement

---

**Format:**

▲[label:]▲LFOR△condition-expression

&lt;statement&gt;

▲[label:]▲NEXT

**Description:**

- The lfor statement repeats a group of statements while the specified condition expression is true.
- An endless loop is formed if “ever” is specified as the condition expression.
- Refer to the syntax diagram for the details concerning the condition expression.

**Note:**

- The generated branch instruction BRA is output as BRAL.

**Example:**

```
lfor [ flag ]  
    jsr  output  
next
```

### LGOTO

LGOTO statement

---

**Format:**

▲[label:]▲LGOTO

**Description:**

- The lgoto statement causes an unconditional jump to any address that is explicitly indicated in the program.
- This statement can be written at any position in the program.
- Use a label to specify the jump address.
- Relative address branch instruction BRAL is generated.

**Example:**

```
    for  [flag1]
      if  [flag2]
        lgoto  LAB1
      endif
      jsr  output
LAB1:
      jsr  inout
    next
```

---

**LIF - (LELSE) - ENDIF**LIF statement

---

**Format:**

```
▲[label:]▲LIF△condition-expression
  <statement>
▲[[label:]▲LELSE]
  <statement>
▲[label:]▲ENDIF
```

**Description:**

- The lif statement changes the flow of control into two directions. The branch direction is determined by the condition expression. The branch is made according to whether the condition expression results in zero (false) or non-zero (true). If true, the immediately following instruction is executed, if false, the instruction following lelse is executed if lelse is coded or instruction following endif is executed if lelse is not coded.
- The lelse part may be omitted.
- There is no limit to the lif statement nesting level.
- When lif statements are nested, the nearest lif and lelse statements are paired.
- Refer to the syntax diagram for the details concerning the condition expression.

**Note:**

- The generated branch instruction BRA is output as BRAL.

**Example:**

```
lif [ flag ]
  [ work ] = 1
lelse
  [ work ] = 2
endif
```

### LLBREAK

LLBREAK statement

---

**Format:**

▲[label:]▲LLBREAK

**Description:**

- The llbreak statement stops execution of the corresponding for, do or switch construct and passes control to the next statement.
- It can be use only within a for, do, or switch construct.
- A relative address branch instruction JMPL is generated.

**Example:**

```
for [ flag1 ]
    if [ flag2 ]
        llbreak
    endif
    jsr  output
next
```



**LLCONTINUE**LLCONTINUE statement

---

**Format:**

▲[label:]▲LLCONTINUE

**Description:**

- The llcontinue statement creates a dummy null statement at the end of the innermost for or do construct containing this statement and passes control to that statement.
- It can be use only within a for or do construct.
- A relative address branch instruction JMPL is generated.

**Example:**

```
for [ flag1 ]
  if [ flag2 ]
    llcontinue
  endif
  jsr  output
next
```

### LLDO - WHILE

LLDO statement

---

**Format:**

▲[label:]▲LLDO

<statement>

▲[label:]▲WHILE△condition-expression

**Description:**

- The lldo statement repeats a group of statements while the specified condition-expression is true. The decision to repeat or not is made after executing the statement. Therefore, the lldo statement is useful in cases where the repeated statements are to be executed once more after the condition is satisfied.
- An endless loop is formed if “ever” is specified as the condition expression.
- Refer to the syntax diagram for the details concerning the condition expression.

**Note:**

- The generated branch instruction BRA is output as JMPL.

**Example:**

```
lldo
    jsr  output
while [ flag ]
```

**LLFOR - NEXT**LLFOR statement

---

**Format:**

▲[label:]▲LLFOR△condition-expression  
    <statement>  
▲[label:]▲NEXT

**Description:**

- The llfor statement repeats a group of statements while the specified condition expression is true.
- An endless loop is formed if “ever” is specified as the condition expression.
- Refer to the syntax diagram for the details concerning the condition expression.

**Note:**

- The generated branch instruction BRA is output as JMPL.

**Example:**

```
llfor [ flag ]  
    jsr  output  
next
```

### LLGOTO

LLGOTO statement

---

**Format:**

▲[label:]▲LLGOTO

**Description:**

- The llgoto statement causes an unconditional jump to any address that is explicitly indicated in the program.
- This statement can be written at any position in the program.
- Use a label to specify the jump address.
- Relative address branch instruction JMPL is generated.

**Example:**

```
    for [flag1]
      if [flag2]
        llgoto LAB1
      endif
      jsr output
LAB1:
      jsr inout
    next
```

---

**LLIF - (LLELSE) - ENDIF**LLIF statement

---

**Format:**

```
▲[label:]▲LLIF△condition-expression
    <statement>
▲[[label:]▲LLELSE]
    <statement>
▲[label:]▲ENDIF
```

**Description:**

- The llif statement changes the flow of control into two directions. The branch direction is determined by the condition expression. The branch is made according to whether the condition expression results in zero (false) or non-zero (true). If true, the immediately following instruction is executed, if false, the instruction following llelse is executed if llelse is coded or instruction following endif is executed if llelse is not coded.
- The llelse part may be omitted.
- There is no limit to the llif statement nesting level.
- When lif statements are nested, the nearest llif and llelse statements are paired.
- Refer to the syntax diagram for the details concerning the condition expression.

**Note:**

- The generated branch instruction BRA is output as JMPL.

**Example:**

```
llif [ flag ]
    [ work ] = 1
llelse
    [ work ] = 2
endif
```

**Format:**

```
▲[label:]▲SWITCH△condition-expression
▲[label:]▲CASE△constant
    <statement>
▲[label:]▲CASE△constant
    <statement>
    :
▲[label:]▲DEFAULT
    <statement>
▲[label:]▲ENDS
```

**Description:**

- The switch statement passes control to one of several statements according to the value of the condition expression. The value of the expression is compared with the case constant at the beginning of the statement and control is passed to the statement that matches. If there is no case with matching value, control is passed to the default statement if it exists. If there is no default statement, switch is exited without executing any statement.
- The default part may be omitted.
- After control is passed to the matching case statement, the subsequent case statements are executed in sequence. The break, lbreak, or llbreak statement can be used to exit the switch construct without executing the next case.
- Refer to the syntax diagram for the details concerning the condition expression and constant.

**Note:**

- Macro arguments cannot be coded in the condition expression of SWITCH or CASE constant.

**Example:**

```
switch [ work ]
  case 1
    jsr  output1
    break
  case 2
    jsr  output2
    break
  case 3
    jsr  output3
    break
  default
    jsr  output4
ends
```

**LSWITCH - CASE - ENDS**

LSWITCH statement

**Format:**

▲[label:]▲LSWITCH△condition-expression

▲[label:]▲CASE△constant

&lt;statement&gt;

▲[label:]▲CASE△constant

&lt;statement&gt;

:

▲[label:]▲DEFAULT

&lt;statement&gt;

▲[label:]▲ENDS

**Description:**

- The lswitch statement passes control to one of several statements according to the value of the condition expression. The value of the expression is compared with the case constant at the beginning of the statement and control is passed to the statement that matches. If there is no case with matching value, control is passed to the default statement if it exists. If there is no default statement, lswitch is exited without executing any statement.
- The default part may be omitted.
- After control is passed to the matching case statement, the subsequent case statements are executed in sequence. The break, lbreak, or llbreak statement can be used to exit the lswitch construct without executing the next case.
- Refer to the syntax diagram for the details concerning the condition expression and constant.

**Note:**

- If lswitch is coded, BRAL is generated by break.
- Macro arguments cannot be coded in the condition expression of SWITCH or CASE constant.

**Example:**

```

lswitch [ work ]
  case 1
    jsr  output1
    break
  case 2
    jsr  output2
    break
  case 3
    jsr  output3
    break
  default
    jsr  output4
ends

```

**Format:**

```
▲[label:]▲LLSWITCH△condition-expression
▲[label:]▲CASE△constant
    <statement>
▲[label:]▲CASE△constant
    <statement>
    :
▲[label:]▲DEFAULT
    <statement>
▲[label:]▲ENDS
```

**Description:**

- The llswitch statement passes control to one of several statements according to the value of the condition expression. The value of the expression is compared with the case constant at the beginning of the statement and control is passed to the statement that matches. If there is no case with matching value, control is passed to the default statement if it exists. If there is no default statement, llswitch is exited without executing any statement.
- The default part may be omitted.
- After control is passed to the matching case statement, the subsequent case statements are executed in sequence. The break, lbreak, or llbreak statement can be used to exit the llswitch construct without executing the next case.
- Refer to the syntax diagram for the details concerning the condition expression and constant.

**Note:**

- If llswitch is coded, JMPL is generated by break.
- Macro arguments cannot be coded in the condition expression of SWITCH or CASE constant.

**Example:**

```
llswitch [ work ]
  case 1
    jsr  output1
    break
  case 2
    jsr  output2
    break
  case 3
    jsr  output3
    break
  default
    jsr  output4
ends
```



**Format:**

▲[label:]▲left-term▲=▲right-term

**Description:**

- The right term is assigned to the left term. Only numeric constant and symbol constant with value of 0 or 1 can be assigned to memory bit variable, register bit variable, and flag variable.
- Refer to the syntax diagram for the details concerning the left and right terms.

**Example:**

```
C = 0 ; Set carry flag to 0.
BIT_A5 = 1 ; Set A register bit 5 to 1.
[ bit0 ] = 0 ; Set memory bit specified by bit0 to 0.
[ bit0 ] = 1 ; Set memory bit specified by bit0 to 1.
X = Y ; Transfer the content of register Y to register X.
[ work ] = 10 ; Transfer 10 to the memory specified by WORK.
[ work ] = [ work1 ] ; Transfer the content of memory specified by
; WORK1 to memory specified by WORK.
```

## B.3 Structured Preprocessor Instruction Syntax Diagram

The syntax of structured instructions available with PRE77 is shown in the form of diagrams.

The following terms are used in the proceeding descriptions.

- Variable

A generic term for memory variable, memory bit variable, register variable, register bit variable, and flag variable.

1. Memory variables

Any memory or stack that is referenced in each 7700 Family addressing mode. They must be enclosed in “[ ]” or “{ }” when coding.

[ZZ]	Direct	[ZZ,X]	Direct X
[ZZ,Y]	Direct Y	[(ZZ)]	Direct indirect
[(ZZ,X)]	Direct indirect X	[(ZZ,Y)]	Direct indirect Y
[HHLL]	Absolute	[HHLL,X]	Absolute X
[HHLL,Y]	Absolute Y	[HHMMLL]	Absolute long
[HHMMLL,X]	Absolute long X	[nn,S]	Stack
[(nn,S),Y]	Stack pointer relative indirect Y		

2. Memory bit variable

Any bit referenced in 7700 Family bit addressing mode. It must be enclosed in “[ ]” or “{ }” when coding. However, this variable must be defined prior to its reference with the pseudo variable .EQU as follows:

Example:

```

BITSYM .EQU 1,1000H ; bit symbol definition
if [ BITSYM ] ; bit symbol reference
:
else
:
endif
    
```

3. Register variables

These variables refer to the various 7700 Family registers. These variables are reserved as register names and should be codes as is. There is no need to enclose these variables in “[ ]” or “{ }”. No distinction is made between uppercase and lowercase characters. Therefore, both A and a are valid.

A	Accumulator A	B	Accumulator B
X	Index register X	Y	Index register Y
S	Stack pointer	PC	Program counter
DT	Data bank register	DPR	Direct page register
PS	Processor status register		

### 4. Register bit variables

Accumulator bits referenced in 7700 Family accumulator bit addressing mode. The following names are reserved by PRE77. No distinction is made between uppercase and lowercase characters. Therefore, both BIT\_A0 and bit\_a0 are valid.

Refer to table 4.1 for the list of reserved words.

### 5. Flag variables

These are the flags in the 7700 Family status register. The following names are reserved by PRE77. No distinction is made between uppercase and lowercase characters. Therefore, both C and c are valid.

C	Carry flag	Z	Zero flag
I	Interrupt disable flag	D	Decimal mode flag
XF	Index register length selection flag	M	Data length selection flag
V	Overflow flag	N	Negative flag
IPL	Processor interrupt priority level		

- WITH\_C

Specifies an operation with carry. This is a reserved word. No distinction is made between uppercase and lowercase characters. Therefore, both WITH\_C and with\_c are valid.

Example:

```
[ work ] = [ work ] << 2 with_c Shift left twice the content of work including carry.
```

- EVER

Specifies an endless loop. This is a reserved word. No distinction is made between uppercase and lowercase characters. Therefore, both EVER and ever are valid.

Example:

```
for ever ; loop endlessly
:
next
```

- Constant

Generic term for numeric constant, character constant, symbol constants, and these combined with instructions.

1. Numeric constant

Represents the number itself.

2. Character constant

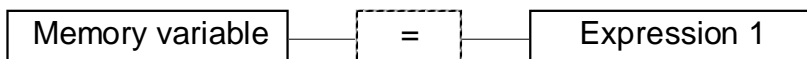
The specified code is treated as ASCII code. It must be enclosed in single or double quotes when coding.

3. Symbol constant

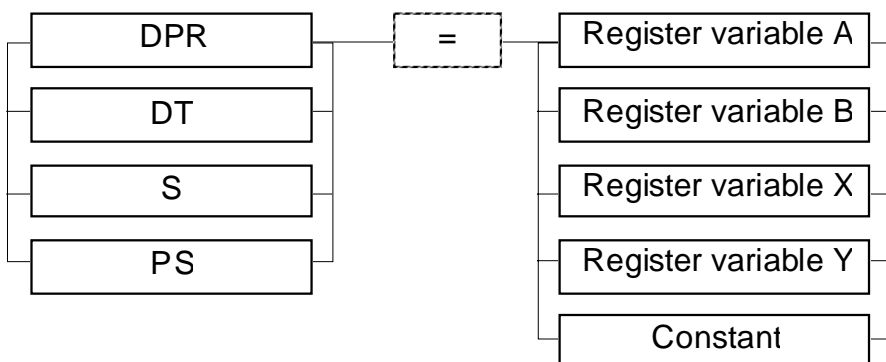
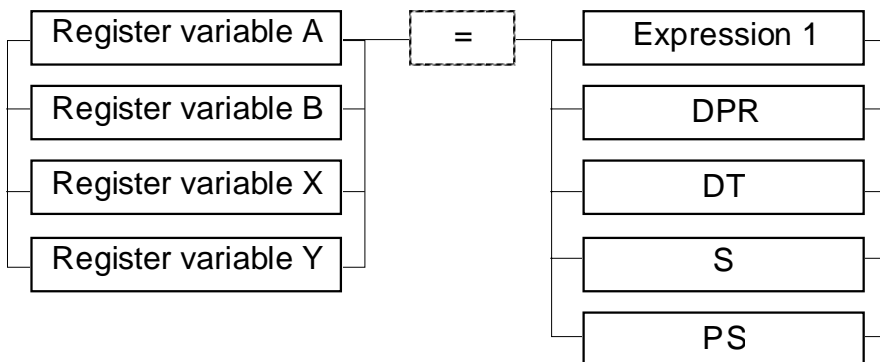
Specifies an alphanumeric character or a special symbol (\*, \_, ?, .). Refer to the syntax diagram for the coding format.

- The coding formats of variables are shown below in the form of syntax diagram.

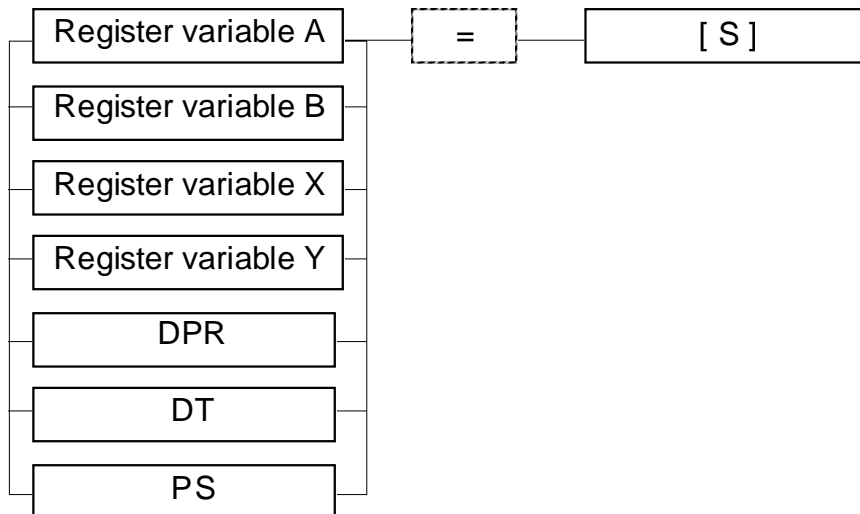
■ Assignment statement



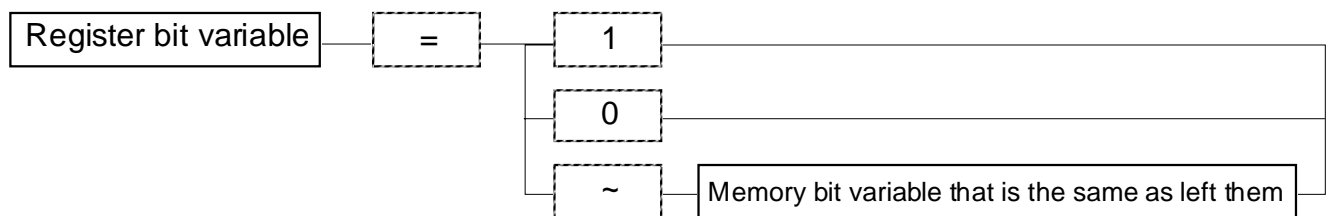
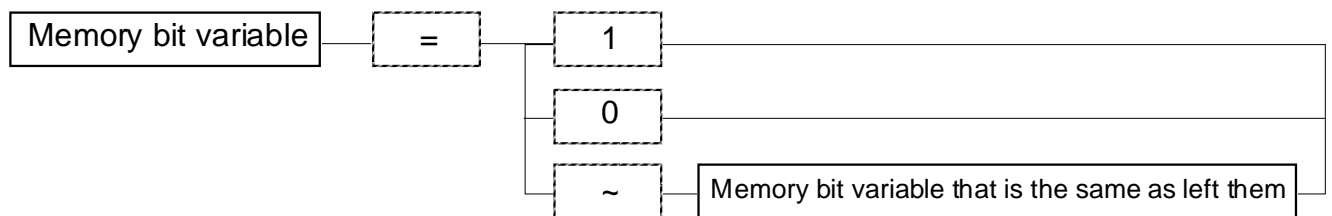
○ Register variable assignment statement



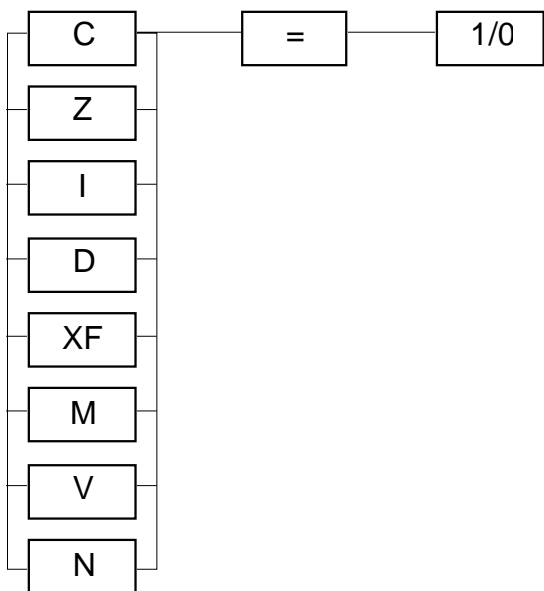
### B.3 Structured Preprocessor Instruction Syntax Diagram



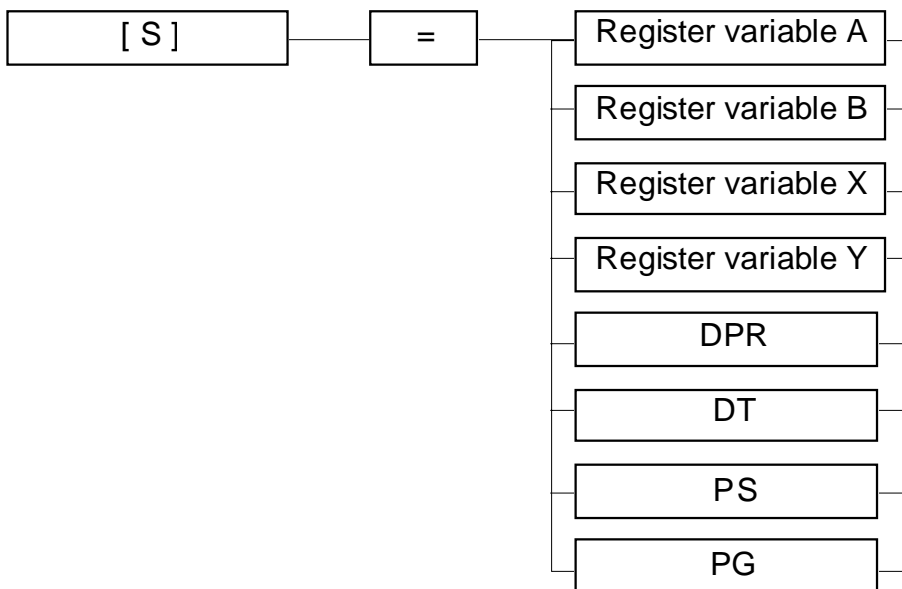
#### ○ Memory bit variable assignment statement



○ Flag variable assignment statement

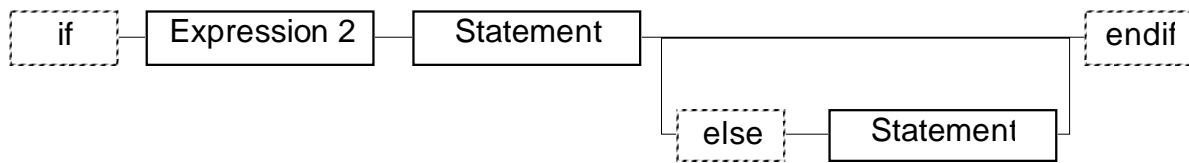


○ Stack frame variable assignment statement

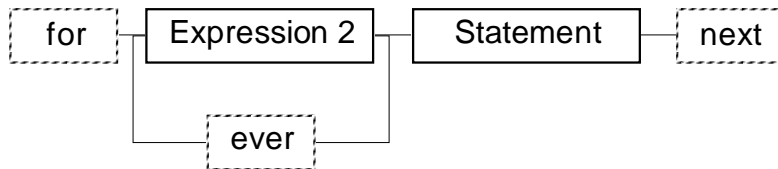


## B.3 Structured Preprocessor Instruction Syntax Diagram

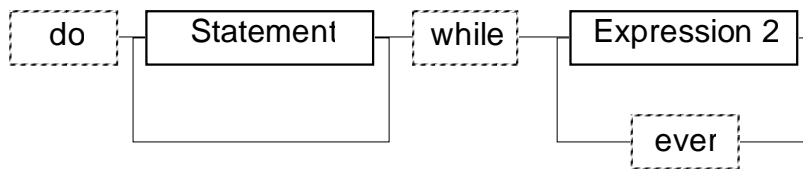
### ■ if statement



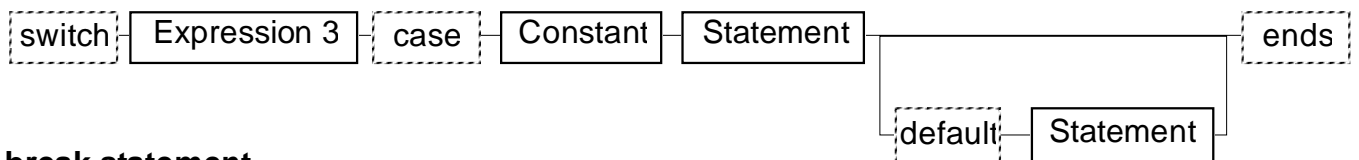
### ■ for statement



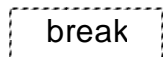
### ■ do statement



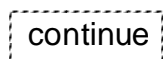
### ■ switch statement



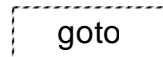
### ■ break statement



### ■ continue statement



### ■ goto statement



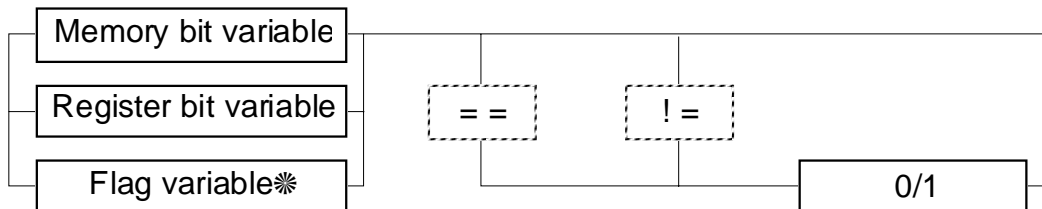
## APPENDIX B. STRUCTURED PREPROCESSOR INSTRUCTIONS

### ■ Expression 1

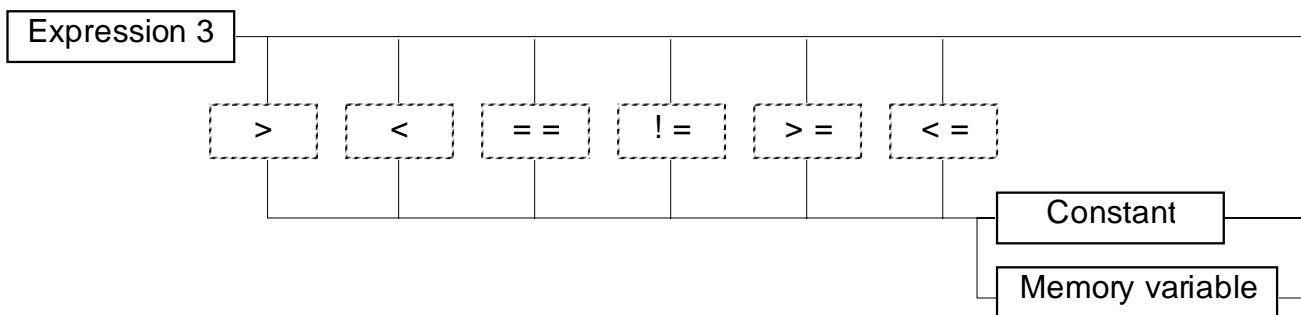
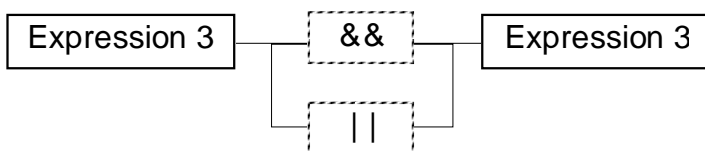
Constant

Expression 3

### ■ Expression 2

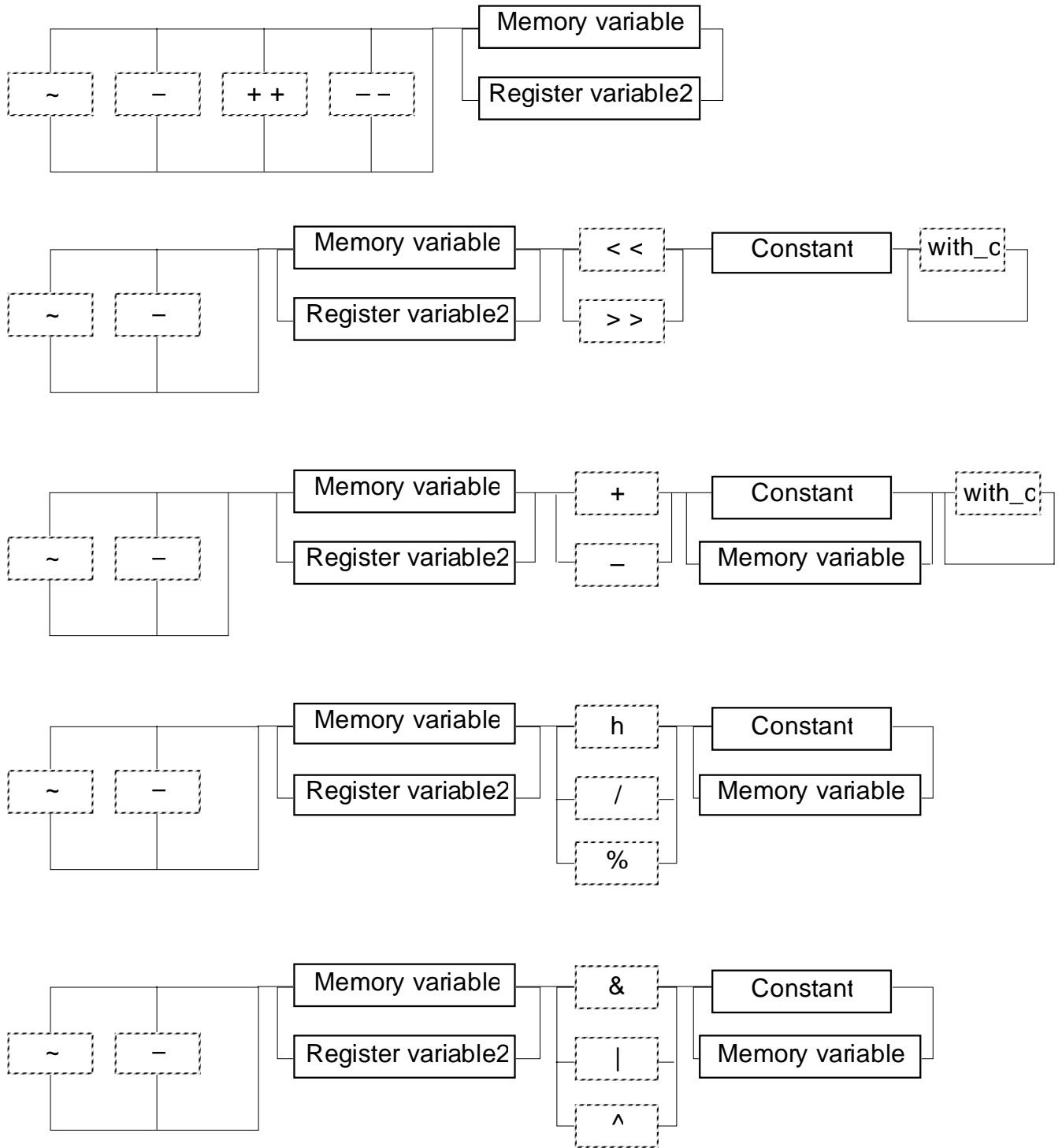


\* 'XF' and 'M' flags are excluded from flag variable.



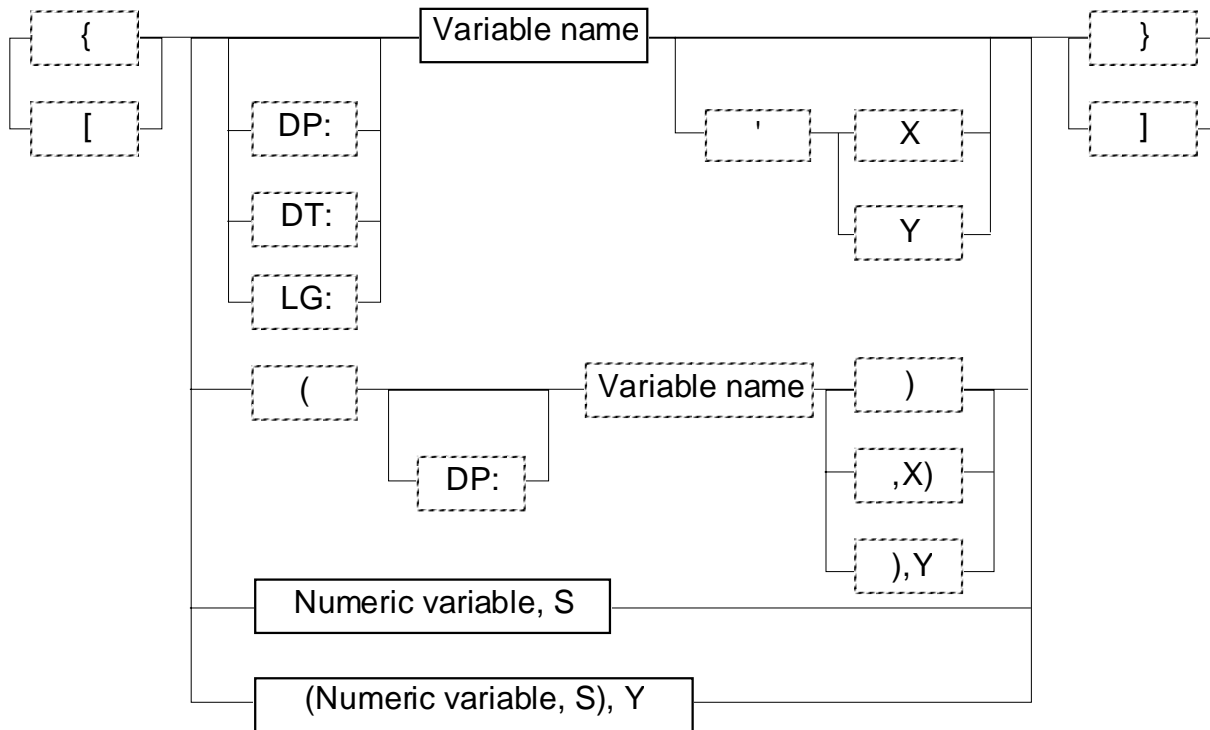


■ Expression 3

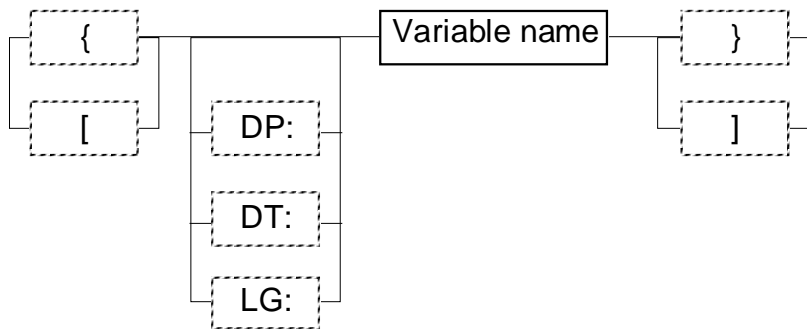


■ Variable

○ Memory variable



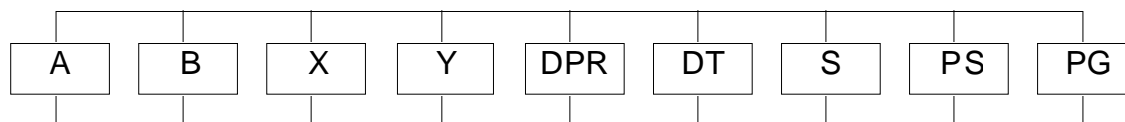
○ Memory bit variable



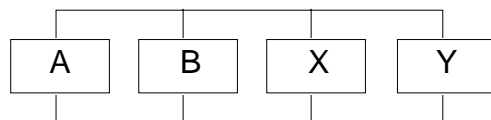
---

### B.3 Structured Preprocessor Instruction Syntax Diagram

○ Register variable



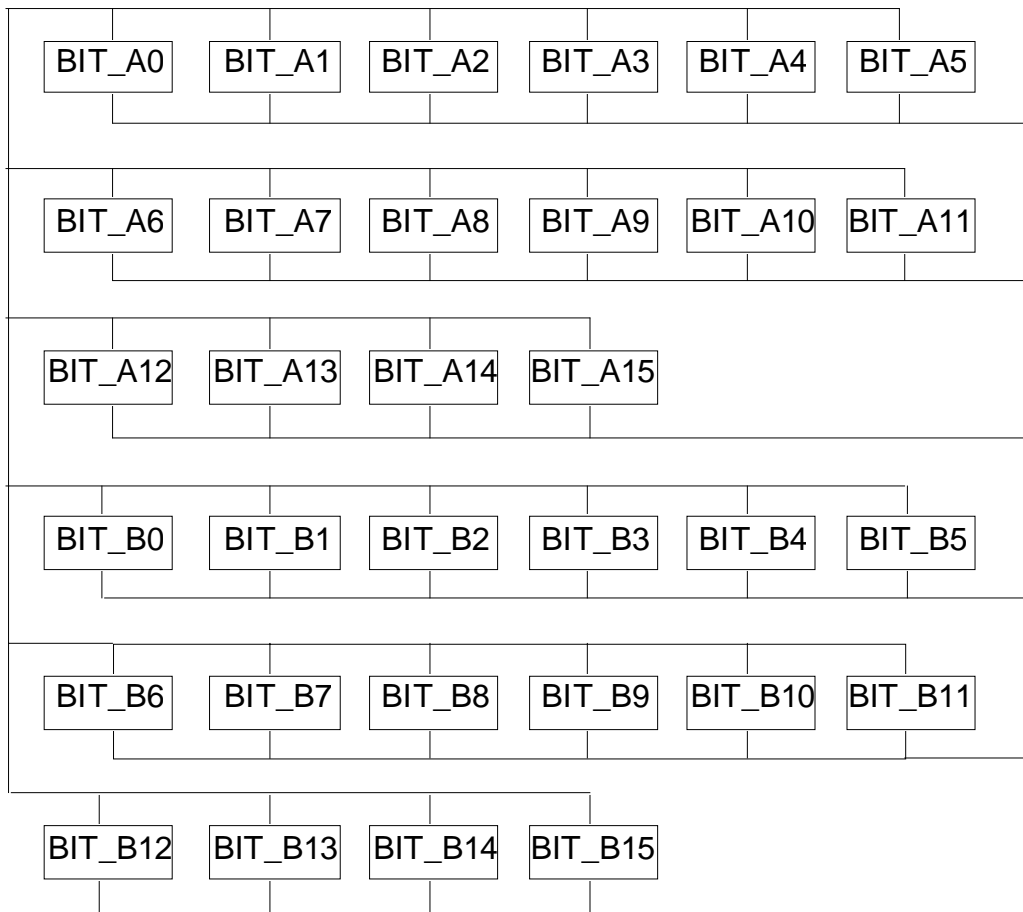
○ Register variable 2



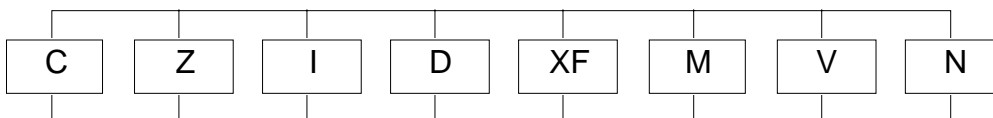
## APPENDIX B. STRUCTURED PREPROCESSOR INSTRUCTIONS

---

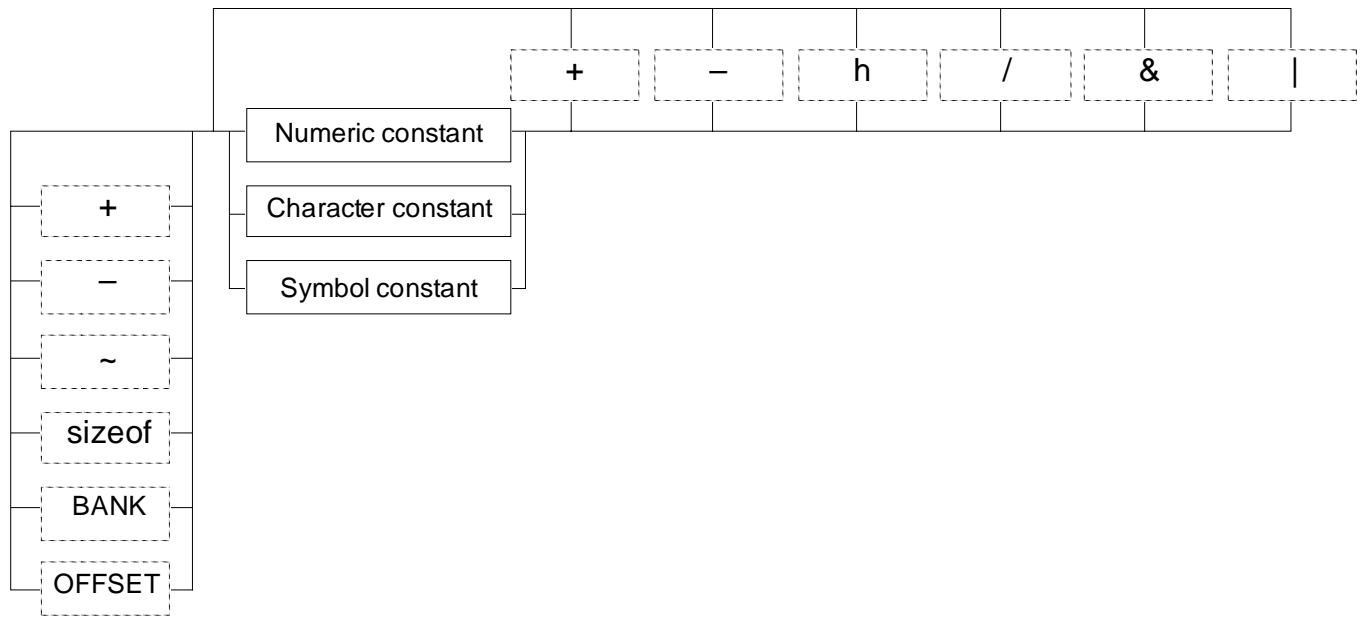
### ○ Register variable



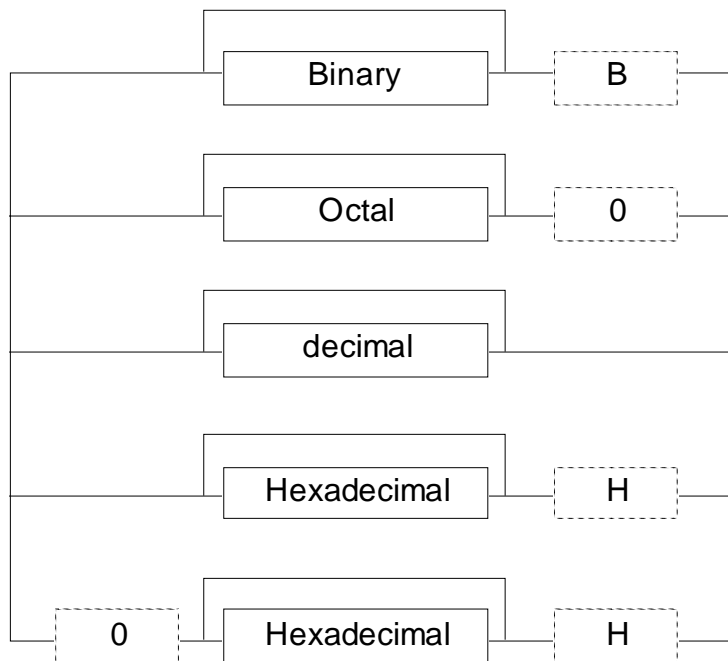
### ○ Flag variable



○ Constant

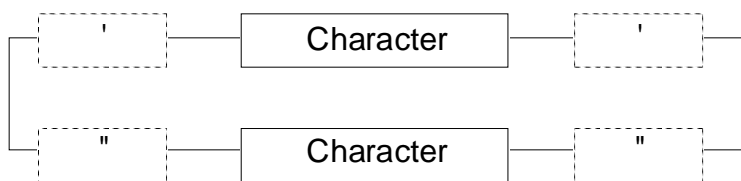


○ Numeric constant

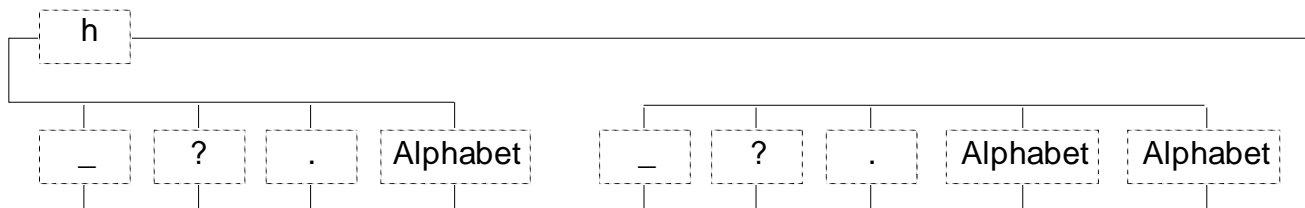


## APPENDIX B. STRUCTURED PREPROCESSOR INSTRUCTIONS

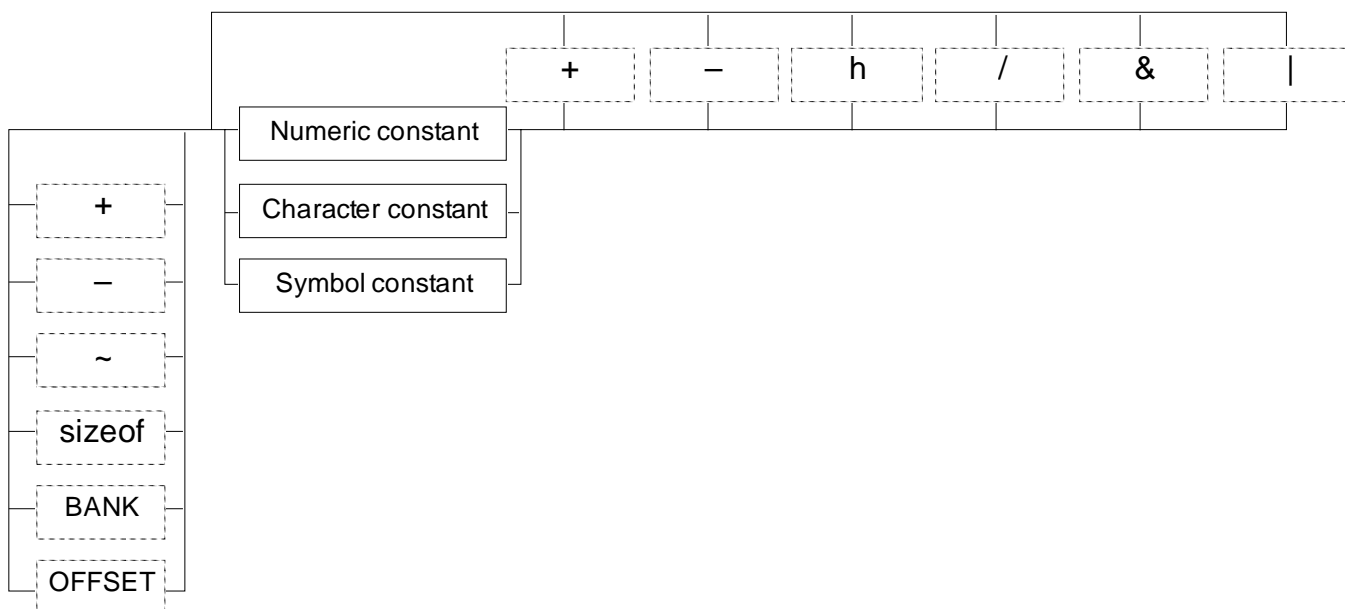
### ○ Character constant



### ○ Symbol constant



### ○ Variable name



# APPENDIX C

## Pseudo Instructions

### C.1 Conventions

The pseudo instructions processed by PRE77 are described in alphabetical order. The following conventions are used in describing each pseudo instruction:

1. Item in [ ] may be omitted.
2. Space or tab code is indicated by  $\triangle$  or  $\blacktriangle$ .  $\triangle$  is a required space or tab code, and  $\blacktriangle$  is an optional space or tab code (i.e., may be omitted).
3.  $\blacktriangle$  is used to separate a label from pseudo instruction. A colon (:) is required when coding a label.

### C.2 Pseudo Instructions

### **.CLINE**

Output line number information

---

**Format:**

▲.CLINE△numeric-value

**Description:**

- Defines the line number information necessary during structured preprocessor source debugging.
- This pseudo instruction is generated automatically by PRE77.

**Note:**

- This pseudo instruction enables source debugging. If this instruction is explicitly coded, a warning is issued and that line is changed to a comment.

### **.DATA**

Declare data length (default is 16)

---

**Format:**

.DATA△expression

**Description:**

- Declares the CPU internal data length (8 or 16). Indicates an 8 bit data if the value of expression is 8; 16 bit data if value of expression is 16.
- This pseudo instruction affects the immediate data length of immediate addressing modes related to the M flag.
- The new data length must be declared with this pseudo instruction when changing data length with the SEM or CLM instruction.
- Note that this pseudo instruction only declares the data length to PRE77 and assembler, and it does not manipulate the data length selection flag (m) for the CPU internal processor status register.

**Note:**

- Macro arguments cannot be coded as the operand of this pseudo instruction.

**Example:**

```
SEM                ; Sets M flag.  
.DATA8             ; Specifies data length.  
A = A + VALUE     ; 8 bit addition
```



---

**.END****Declares the end of program**

---

**Format:****▲.END****Description:**

- This pseudo instruction specifies the end of the source program.
- Lines after this pseudo instruction are not processed.

**Example:**

```
.END ; Declares the end of program
```

---

**.ENDFUNC****Declares the end of structure preprocessor program**

---

**Format:****▲.ENDFUNC△label****Description:**

- This pseudo instruction declares the end of structured preprocessor program.
- This instruction enables source line debugging.
- This pseudo instruction is generated automatically by PRE77.

**Note:**

- This pseudo instruction enables source debugging. If this instruction is explicitly coded, a warning is issued and that line is made into a comment.

**.EQU****Equation**

---

**Format 1:**

symbol△.EQU△expression

**Format 2:**

bit symbol△.EQU△expression,expression

**Description:**

- Sets a numeric value to a symbol.
- Symbols used in the expression must be defined before this line.
- Format 1 assigns a double word numeric value to symbol. Format 2 assigns a 0 - 15 bit value and an address to symbol.

**Notes:**

- Only already defined symbols can be used in expression. Labels are not allowed.
- Macro arguments cannot be coded as the operand of this pseudo instruction.

**Example:**

```
sym      .EQU      1000H    ; Assign sym to 01000H
flag     .EQU      0,100H   ; Assign flag to bit 0 of 100H
```

**.FUNC****Start structured preprocessor program**

---

**Format:**

▲.FUNC△label

**Description:**

- This pseudo instruction declares the start of structured preprocessor program.
- This instruction enables source line debugging.
- This pseudo instruction is generated automatically by PRE77.

**Note:**

- This pseudo instruction enables source debugging. If this instruction is explicitly coded, a warning is issued and that line is made into a comment.

---

**.INCLUDE****Load a file**

---

**Format:****▲.INCLUDE△filename****Description:**

- This pseudo instruction loads the structured preprocessor program file specified with the operand at the point where this pseudo instruction is coded.
- The filename must be specified in full.
- This pseudo instruction can be nested up to 9 levels.

**Example:**

```
.INCLUDE    TEST.P77    ; Load the content of TEST.P77
```

### **.INDEX**

Declare index register length (default is 16)

---

**Format:**

▲.INDEX△expression

**Description:**

- Declares the CPU internal index register length (8 or 16).
- Indicates an 8 bit data if the value of expression is 8; 16 bit data if value of expression is 16.
- The index register length must be declared with this pseudo instruction when changing the index register length with “CLM X” or “SEP X” instruction.

**Note:**

- Note that this pseudo instruction only declares the index register length to the assembler, and it does not manipulate the index register length selection flag (m) of the CPU internal processor status register.
- Macro arguments cannot be coded as the operand of this pseudo instruction.

**Example:**

```
SEP      X      ; Set the X flag
.INDEX   8      ; Specify the index register length
X = VALUE ; 8 bit load
```

**.SECTION**

Declare section name

**Format:**

▲.SECTION△section name

**Description:**

- PRE77 recognizes lines following this pseudo instruction as structured preprocessor program.
- Any name can be specified as section name. More than one section in the same file can have the same name.

**Note:**

- This pseudo instruction must be coded at the beginning of the program (an error will occur if it is missing).
- Do not write this pseudo-instruction in the processing line of RAMS77's conditional assemble pseudo-instruction ".IF." Since conditional assemble is not processed by PRE77, an error result if the assembly source that was generated after adding an option to output source line information is processed by RASM77.

**Example:**

```

        .SECTION    DATA    ; Start DATA section
tabletop:
        .BYTE      'ABCDEFGH'
        :
        .SECTION    PROG    ; Start PROG section
_init:
        .DATA          16
        .INDEX         16
        X = 0
        for X <= 20
            [PORT0] = tabletop,X
            X = X+1
        next

```

### **.SOURCE**

Define source file name

---

**Format:**

▲.SOURCE△source-filename

**Description:**

- This pseudo instruction defines the file name necessary for source debugging.
- This pseudo instruction is generated automatically by PRE77.

**Note:**

- This pseudo instruction enables source debugging. If this instruction is explicitly coded, a warning is issued and that line is made into a comment.

PART 3

LINKAGE EDITOR FOR  
7700 FAMILY



**LINK77 OPERATION MANUAL**

---

# Table of Contents

## Chapter 1. Organization of LINK77 Operation Manual

### Chapter 2. Overview

2.1 Functions .....	2
2.2 Files Created .....	2
2.3 Organization of MAP File .....	3

### Chapter 3. Function of Section

3.1 Purpose of Sections .....	5
3.2 Attributes of Sections .....	6
3.2.1 Address Attribute .....	7
3.2.2 Physical Attributes .....	7
3.3 Basic Functions of Sections .....	7

### Chapter 4. Operation

4.1 Starting LINK77 .....	8
4.2 Input Parameters .....	8
4.2.1 Relocatable Filename .....	8
4.2.2 Library Filename .....	9
4.2.3 Section Control .....	9
4.2.4 Command Parameters .....	10
4.3 Input Modes .....	11
4.3.1 Prompt Mode .....	11
4.3.2 Command Line Input Mode .....	13
4.3.3 Command File Input Mode .....	14
4.4 Errors .....	15
4.4.1 Error Types .....	15
4.4.2 Return Values to MS-DOS .....	16
4.5 Environment Variables .....	16

### Appendix A. Error Messages

A.1 List of Link Errors .....	16
A.2 Warning Message .....	22



---

## Appendix B. Original HEX Format for 7700 Family

<b>B.1 Original HEX Format for 7700 Family .....</b>	<b>23</b>
<b>B.2 HEXTOS2 .....</b>	<b>24</b>
B.2.1 Overview .....	24
B.2.2 Function .....	24

---

## List of Figures

Figure 2.1 MAP File Output Example .....	4
Figure 3.1 Configuration of Relocatable File .....	5
Figure 3.2 System Memory Map .....	6
Figure 4.1 LINK77 Starting Screen .....	12
Figure 4.2 Screen for Prompt Mode Input .....	12
Figure 4.3 Example of Command Line Input - 1 .....	13
Figure 4.4 Example of Command Line Input - 2 (Parameters Omitted) ..	13
Figure 4.5 Example of Command Line Input - 3 (Command Parameters Omitted) .....	13
Figure 4.6 Specifying a Command File .....	14
Figure 4.7 Example of Command File Specification .....	14
Figure 4.8 Example of Error Display .....	15

---

## List of Tables

Table 4.1 List of Command Parameters .....	10
Table 4.2 Restricted Branch Instructions .....	11
Table 4.3 List of Error Levels .....	16
Table A.1 List of Link Error Messages .....	18
Table A.2 List of Warning Message .....	22

---

# CHAPTER 1

## Organization of LINK77 Operation Manual

The LINK77 Operation Manual consists of the following chapters:

- Chapter 2. Overview  
Describes the basic functions of the LINK77 linkage editor and the files created by LINK77.
- Chapter 3. Function of Section  
Describes section which is the basic unit in which LINK77 manipulates programs.
- Chapter 4. Operation  
Explains how to enter LINK77 commands.
- Appendix A. Error Messages  
Lists the messages output by LINK77 along with explanation of the errors and the actions to be taken.
- Appendix B. Original HEX Format for 7700 Family  
Describes the original HEX format that is used to output 1 M bytes or larger machine language data for 7700 Family.

# CHAPTER 2

## Overview

The LINK77 linkage editor links relocatable files created by the RASM77 relocatable assembler with library files, and creates a 7700 Family machine language data file.

### 2.1 Functions

LINK77 can be used together with LIB77<sup>1</sup>. To maximize the utility of these software products, LINK77 offers the following functions:

1. Places the sections of different relocatable files that have the same section name at contiguous locations in the linked file.
2. The sequence in which sections<sup>2</sup> are to be placed in the linked file can be specified. Also, the starting address for each section can be specified.
3. Library files created by LIB77 can be used.
4. Machine language data for the entire (16M byte) memory space of 7700 Family can be generated.
5. Map file which is useful for debugging is created.
6. Symbolic file that is necessary for symbolic debugging by debugger is created.

### 2.2 Files Created

LINK77 creates three types of files:

1. Machine language data file (hereafter referred to as HEX file)
  - HEX file is output in the extended Intel hexadecimal format when the address range is within 1 M bytes. When the address range exceeds 1 M bytes, HEX file is output in the original 7700 Family HEX format<sup>3</sup>.

---

<sup>1</sup> LIB77 is the name of the 7700 Family librarian program.

<sup>2</sup> Sections are the basic units that make up a program. Physically different units such as ROM areas and RAM areas are sections. For more detailed explanation, see Chapter 4.

<sup>3</sup> For the specifications of the original HEX format for 7700 Family, see Appendix B.

- The output format is automatically selected based on the address range size.
  - HEX files have the extension, .HEX.
2. Mapping file (hereafter referred to as MAP file)
    - MAP file contains the final location information for the sections of files that have been linked.
    - MAP file can be printed for use in debugging and for determining the memory size of each section.
    - MAP file is output when the command parameter “-M” is specified.
    - MAP file has the extension, MAP.
    - Organization of MAP file is described in the next section.
  3. Symbolic file (hereafter referred to as SYM file)
    - SYM file contains various information necessary for symbolic debugging.
    - SYM file is output when the command parameter “-S” is specified.
    - SYM file has the extension, .SYM.

The above output file names (excluding extension) can be specified with the command option “-F”.

## 2.3 Organization of MAP File

Figure 2.1 is a sample printout of a MAP file. A MAP file includes the following information:

1. Information, for each section, concerning how much data has been linked from which relocatable file. This portion of MAP file contains the following:
  - ATR: Specifies either relative or absolute<sup>5</sup> attribute. REL specifies relocatable attribute, and ABS specifies absolute attribute.
  - TYPE: Specifies either RAM or ROM area.
  - START: Indicates the start address.
  - LENGTH: Specifies the area size in bytes.
  - ALIGNMENT: Specifies “WORD” if word alignment<sup>6</sup> was performed during linkage.
  - When a library file has been linked, both library filename and relocatable filename are shown. Relocatable filename is indicated in parentheses.

---

<sup>5</sup> An assembly language source file in which starting address is specified by the pseudo instruction .ORG has the absolute attribute.

<sup>6</sup> Word alignment is performed when the “-W” parameter is specified for a linkage command.

## CHAPTER 2. OVERVIEW

---

### 2. Global labels table

Global labels table lists the global labels<sup>7</sup> used in the program along with their absolute addresses. This portion of the MAP file is output only when the command parameter “-MS” is specified.

### 3. Global symbols table

Global symbols table lists the global symbols<sup>8</sup> used in the program along with their absolute addresses. This portion of the MAP file is output only when the command parameter “-MS” is specified.

```
7700 Family LINKER V.2.02.10      MAP FILE                      Thu Jul 31 10:41:53 1997

SECTION          FILENAME          ATR.  TYPE  START  LENGTH  ALIGNMENT

WORK             MAIN.R77          ABS   RAM  000000  000080
                SUB.R77          REL   RAM  000080  000100
                UTIL.LIB        REL   RAM  000180  000008
                (CALC.R77)
PR0M             MAIN.R77          REL   ROM  00C000  001800
                SUB.R77          REL   ROM  00D800  001500
                UTIL.LIB        REL   ROM  00ED00  000820
                (CALC.R77)
DR0M             MAIN.R77          ASS   ROM  00F520  000023  WORD
                SUB.R77          REL   ROM  00F544  000030

GLOBAL LABEL INFORMATION

ACNT 000030     COUNT                00009C  DATA0      0000A4
DATA1 0000A6     MAIN                 00C000  TIME        0000C6

GLOBAL SYMBOL INFORMATION
```

**Figure 2.1 MAP File Output Example**

---

<sup>7</sup> Global label is a label that has been defined in another file.

<sup>8</sup> Global symbol is a label that has been defined by the pseudo instruction .EQU in another file

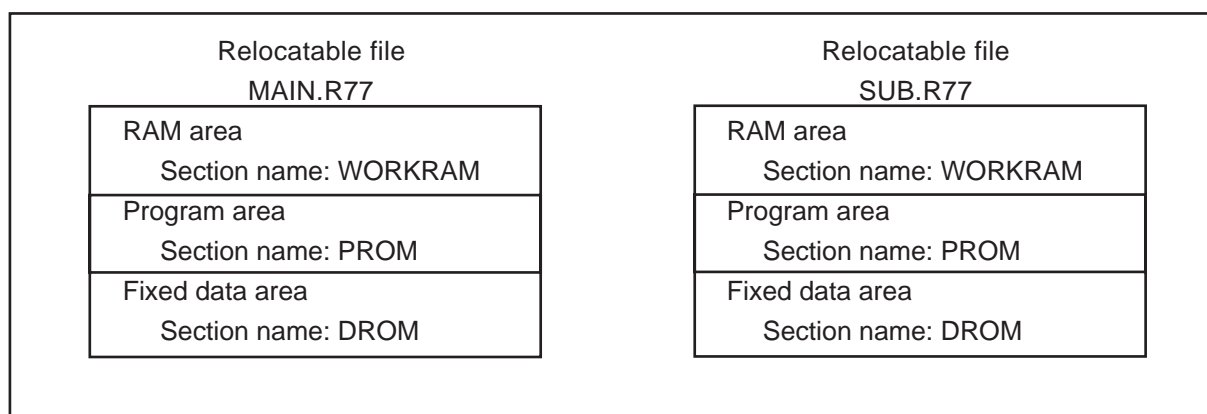
# CHAPTER 3

## Function of Section

### 3.1 Purpose of Sections

A program written in assembly language typically consists of a RAM area, program area and fixed data area. When RASM77 assembles a source file, it creates a relocatable file which contains at least one of these areas. Each of these areas is called a section. The contents and purpose of sections are described below by referring to specific examples.

Two relocatable files shown in Figure 3.1, MAIN.R77 and SUB.R77, provide the most clear examples of sections. Each of these relocatable files has a RAM area, program area and fixed data area.



**Figure 3.1 Configuration of Relocatable File**

To arrange these relocatable files in the memory space as illustrated in Figure 3.2, the sections to be linked together must be assigned an identical section name in advance with the pseudo instruction SECTION. Then, the sections with identical section name are placed in a contiguous area by the linkage process. LINK77 allows specification of the starting address for each section at the time of linkage.



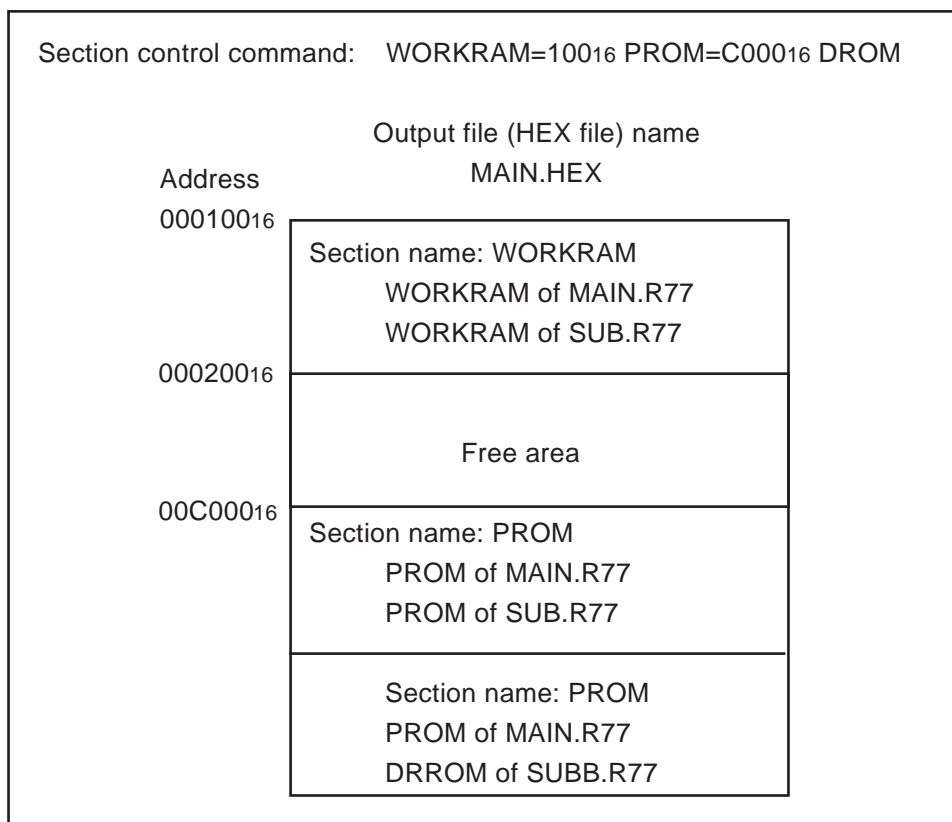


Figure 3.2 System Memory Map

In this way, LINK77 commands enable generation of machine language data corresponding to the final locations on the user system.

### 3.2 Attributes of Sections

Each section has two types of basic information called address attribute and physical attribute. These attributes are explained in details below.

### 3.2.1 Address Attribute

Address attribute of a section is determined by whether or not the pseudo instruction `.ORG` is specified in the section in the source program. There are two types of location attribute:

1. Relocatable attribute
  - A section that does not include the pseudo instruction `.ORG` has the relocatable attribute.
2. Absolute attribute
  - A section that includes the pseudo instruction `.ORG` has the absolute attribute.
  - Starting address cannot be specified for a section with the absolute attribute at the time of linkage.
3. Sections having an identical section name but existing in different relocatable files can have different address attributes.

### 3.2.2 Physical Attributes

Physical attribute specifies the physical property of the area where a section is placed. There are two types of physical attribute:

1. ROM attribute
  - A section with the ROM attribute is converted to a HEX file by linkage editing.
  - A section of assembly language source file that contains a code generating statement has the ROM attribute.
  - To avoid confusion with address attribute, ROM attribute is referred to as ROM type hereafter.
2. RAM attribute
  - A section with the RAM attribute is not converted to a HEX file by linkage editing.
  - A section of assembly language source file that contains the area allocation pseudo instruction `.BLKB`, `.BLKW`, etc. has the RAM attribute.
  - To avoid confusion with address attribute, RAM attribute is referred to as RAM type hereafter.
3. All sections that have the identical section name must have the same physical attribute.

## 3.3 Basic Functions of Sections

1. Sections having the identical section name are placed in contiguous locations by linkage editing. Sections with different section names are never placed between sections with identical section name.
2. Within a group of sections with the identical section name, sections are placed in the order in which relocatable files are specified in the linkage command.

# CHAPTER 4

## Operation

### 4.1 Starting LINK77

To execute LINK77, the following information (input parameters) must be input:

1. Relocatable filenames (required)
2. Library filenames
3. Section control information
4. Command parameters

Input parameters for LINK77 execution can be input in any of the following three modes depending on the operating environment:

1. Prompt mode
2. Command line input mode
3. Command file input mode

Same input parameters are used for the three input modes available. Also, same commands can be executed regardless of the input mode being used. The input parameters are explained in Section 4.2, and each input mode is explained in detail by referring to examples in Section 4.3.

### 4.2 Input Parameters

#### 4.2.1 Relocatable Filename

1. Relocatable filenames must always be input.
2. LINK77 processes only the relocatable files with the extension, .R77. The filename extension can be omitted during command input.
3. Filename may be specified with a directory path. If only filename is specified, LINK77 processes a file in the current drive's current directory.

4. Name of the first relocatable file specified is used as the filename of the output file. If the “-F” parameter is used to specify the filename, that filename becomes the output file.
5. Output file is output to the directory in which the first relocatable file specified resides. If the output file is specified by the command parameter “-O”, the command parameter specification is used.

### 4.2.2 Library Filename

1. Specification of library filenames may be omitted.
2. LINK77 processes only library files with the extension, .LIB. The filename extension can be omitted during command input.
3. Library filename may be specified with a path name. If only the filename is specified, LINK77 processes a file in the current directory.
4. Library files are referenced during linkage editing only if there are global labels or symbols that cannot be resolved in the relocatable files.

### 4.2.3 Section Control

1. Specification of section control information may be omitted. If section control information specification is omitted, sections are placed in the linked file in the order in which they were found in the relocatable files.
2. Section control information should be specified only for relocatable section. Absolute sections are placed at fixed address specified with the pseudo instruction .ORG regardless of the section specification.
3. Section placement sequence, if specified, must be specified starting from the lower-order address, using a space as the delimiter between section names.
4. Starting address for each section, if specified, must be specified in the “section name=address” format. Address must be specified in hexadecimal (beginning ‘0’ and ending ‘H’ need not be specified).
5. If the start address of a relocatable section is omitted, it is placed starting from address 0.
6. If section control information is not specified, each relocatable section is placed immediately after its preceding section. If, however, word alignment is specified with the command parameter “-W”, relocatable section is placed at word boundary.
7. LINK77 recognizes uppercase and lowercase characters in section name as different characters.

## CHAPTER 4. OPERATION

---

8. Overlapping of section addresses causes an error.

However, if the command parameter “-A” is specified, absolute addresses may be overlapped. This allows more than one program file to be loaded with the .INCLUDE pseudo instruction without specifying external reference for absolute address label in SFR area.

### 4.2.4 Command Parameters

Command parameters are used to control linkage output file, version check, word alignment, etc. Table 4.1 lists the command parameters available with LINK77.

**Table 4.1 List of Command Parameters**

Command parameter	Description
-A	Allows absolute sections with same name to overlap. This can be used to combine shared global memory areas.
-C	Issues a warning when a certain branch instruction <sup>1</sup> is on bank boundary. A warning is also issued when there is a data that has the same value as the machine word of the instruction.
-BRAL	LINK77 outputs warning when processing with this option if the code data which the jump address of BRAL is 0xFFFFH is found.
-F	Specifies the output file name. The format is as follows: -FTEST Output files are output with the file names TEST.HEX, TEST.MAP, and TEST.SYM.
-M	Outputs a MAP file. (Section information only)
-MS	Outputs a MAP file including global labels and symbols listings.
-N	Ignores the .R77 and .LIB file reference information specified with the pseudo instructions .OBJ and .LIB in the source file.
-O	Specifies the directory for the output file. The format is as follows: -OC:\USR\WORK Specifies to output the output file to the \USR\WORK directory on drive C.
-S	Outputs a SYM file.
-V	Checks for version consistency between relocatable files. To use this function, version declaration must be made in the assembly language source files using the pseudo instruction .VER.
-W	Aligns section on word boundary.

## Notes

1. The Series 7700 Family has a restriction which causes a branch to be made to the address within the next bank rather than to the address specified in the program when certain branch instructions are allocated at the highest address of each bank or spans across banks. Table 4.2 lists the instructions for which this restriction applies.

Table 4.2 Restricted Branch Instructions

Instruction	Addressing Mode	Bytes	Machine Code
RTS	Implied	1	60 <sub>16</sub>
JMP	Absolute	3	4C <sub>16</sub>
	Absolute indirect	3	6C <sub>16</sub>
	Absolute indexed X indirect	3	7C <sub>16</sub>
	Absolute indirect long	3	DC <sub>16</sub>
JSR	Absolute	3	20 <sub>16</sub>
	Absolute indexed X indirect	3	FC <sub>16</sub>

2. Word alignment

Word alignment is the alignment of section starting addresses on word boundary. Word alignment increases the memory size, but it can speed up execution because section starting instructions are always read as words. (Not effective when bus width is 1 byte.)

If word alignment causes a 1 byte space between sections and the section at the higher-order address is ROM type, a NOP instruction (0EAH) is written at the beginning of the ROM area.

## 4.3 Input Modes

### 4.3.1 Prompt Mode

The prompt mode has the following features:

1. This input mode allows interactive input of relocatable filenames, library filenames, section control commands and command parameters, in this order.
2. This input mode is convenient when there are only few relocatable files and sections to be linkage edited. It is also convenient for setting address by trial and error.

## CHAPTER 4. OPERATION

---

- LINK77 automatically switches to the prompt mode when the necessary commands are missing in the input command line or command file.

The prompt mode is started by entering LINK77 <RET> from the MS-DOS prompt. LINK77 outputs the following messages to the screen when the prompt mode is started:

```
A>LINK77 <RET>
7700 Family LINKER V.2.02.10
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Relocatable files (.R77) >>
```

**Figure 4.1 LINK77 Starting Screen**

The last line in Figure 4.1 shows that LINK77 is waiting for input of relocatable filenames. Names of the relocatable files to be linked must be entered to the right of ». LINK77 then waits for entry of library filenames, followed by section control information and command parameters, in this order. Entries should be made as illustrated in Figure 4.2.

```
A>LINK77 <RET>
7700 Family LINKER V.2.02.10
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Relocatable files (.R77) >> MAIN SUB<RET>
Libraries          (.LIB) >> UTIL1 UTIL2<RET>
Section information >> WORKRAM=100 PROM=C000 DROM<RET>
Command parameter >> -O\WORK -M -S<RET>
```

**Figure 4.2 Screen for Prompt Mode Input**

### 4.3.2 Command Line Input Mode

The command line input mode has the following features:

1. This input mode allows input of all linkage editing commands for the MS-DOS command prompt.
2. Because MS-DOS limits command length to no more than 127 characters, this input mode should be used when there are only few relocatable files and sections to be link edited.
3. This input mode can be also used when specifying execution commands in a batch file or a make file.
4. The four types of input parameter information must be entered using comma (,) as delimiter. Figure 4.3 shows command line mode input of the same commands as shown in Figure 4.2.
5. Comma must be entered even when there are library filename and subsequent parameters. If no library files are required, the example in Figure 4.3 would change to that shown in Figure 4.4.
6. In a special case where command parameters are omitted, two commas must be entered to clearly specify that there are no command parameters. If command parameters are omitted in the example in Figure 4.4, it would change to that shown in Figure 4.5.
7. If required input parameters are missing, LINK77 switches automatically to the prompt mode.

```
A>LINK77 MAIN SUB, UTIL1 UTIL2, WORKRAM=100 PROM=C000 DROM, -O\WORK -M -S<RET>
```

**Figure 4.3 Example of Command Line Input - 1**

```
A>LINK77 MAIN SUB,, WORKRAM=100 PROM=C000 DROM, -O\WORK -M -S<RET>
```

**Figure 4.4 Example of Command Line Input - 2 (Parameters Omitted)**

```
A>LINK77 MAIN SUB,, WORKRAM=100 PROM=C000 DROM, ,<RET>
```

**Figure 4.5 Example of Command Line Input - 3 (Command Parameters Omitted)**



### 4.3.3 Command File Input Mode

The command file input mode has the following features:

1. In this input mode, linkage commands are created in advance in a command file using an editor, and the name of this command file is specified when starting LINK77.
2. The command file input mode is convenient when the command line input mode cannot be used because there are too many characters in the commands to be input.
3. The command file input mode can also be used when specifying execution commands in a batch file or a make file.
4. Command filename is entered using @ as prefix when starting LINK77 as shown in Figure 4.6. In the example shown in Figure 4.6, the contents of the file, CMD.DAT are executed as commands.
5. Commands can be specified in a command file in the same manner as when using the command line input mode (except that "LINK77" need not be specified to start LINK77). Line feed code is ignored so that a long command can be specified on multiple lines. The commands in the example shown in Figure 4.5 can be created in a command file illustrated in Figure 4.7.
6. When a required command parameter is missing, LINK77 automatically switches to the prompt mode. For example, if the second comma in the last line of the example in Figure 4.7 is missing, LINK77 switches to the prompt mode screen and prompts for command parameter input.

```
A>LINK77 @CMD.DAT<RET>
```

**Figure 4.6 Specifying a Command File**

```
MAIN SUB  
,  
,WORKRAM=100 PROM=C000 DROM  
,,
```

**Figure 4.7 Example of Command File Specification**

## 4.4 Errors

### 4.4.1 Error Types

The following types of errors may occur during execution of LINK77:

1. OS errors

Errors related to the environment in which LINK77 is executed. These errors include disk and memory shortages. When such an error occurs, the error message list in Appendix A should be checked and the appropriate OS command should be entered.

2. LINK77 command line input errors

These are the errors in the LINK77 startup command line. The input command should be checked against the descriptions in this chapter, and a correct command line must be re-entered.

3. Errors in relocatable files to be linked

These are errors in the contents of the relocatable files being linked such as duplicate global label definitions and referencing of undefined symbols. The source files must be checked and re-assembled if necessary.

4. LINK77 function errors

These are the errors caused by use of different versions of the LINK77, RASM77 and LIB77 programs, for example. If the cause of error cannot be determined, contact Mitsubishi Electric Semiconductor Software Corporation.

When LINK77 detects an error, it outputs error information in the format shown in Figure 4.8. The information should be checked against the error message list (in error number order) in Appendix A, and appropriate action must be taken.

```
A>LINK77 MAIN SUB,,WORKRAM=100 PROM=C000 DROM,,  
7700 Family LINKER V.2.02.10  
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION  
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION  
All Rights Reserved.  
  
now processing pass 1  
processing "MAIN.R77"  
ERROR NO.2: Out of heap space  
  
A>
```

**Figure 4.8 Example of Error Display**

### 4.4.2 Return Values to OS

When using an OS batch execution file, there are times when it is desirable to modify processing according to the results of execution. LINK77 returns one of five error levels to OS depending on the result of execution as summarized in Table 4.3. For explanation of how to utilize these error levels, refer to an OS reference guide .

**Table 4.3 List of Error Levels**

Error level	Execution result
0	Normal termination
1	Error in contents of relocatable files to be linked
2	LINK77 command input error
3	MS-DOS error
4	LINK77 function error

## 4.5 Environment Variables

The environment variable "LIB77" can be used when LINK77 searches library files. When the directory of the library file is specified with the environment variable "LIB77", no directory needs to be specified when starting LINK77. The following example specifies the directory USR on drive A as the library reference directory.

Example:     A>SET LIB77=A:\USER<RET>

# APPENDIX A

## Error Messages

### A.1 List of Link Errors

## APPENDIX A. ERROR MESSAGES

---

**Table A.1 List of Link Error Messages**

<b>Error No.</b>	<b>Error message</b>	<b>Meaning and actions</b>
0	xxx file not found	Input file cannot be found. (Includes a file specified by the pseudo instruction .OBJ or .LIB.) ⇒ Reenter the filename correctly.
1	Invalid command input	Five or more parameters are specified in input command, or 2048 or more characters are specified in input command. ⇒ Reenter the input command with no more than four parameters and 2048 characters.
2	Out of heap space	Memory space is insufficient to execute the linker. ⇒ Reduce the number of public symbols.
3	Invalid section information	Section information specification is invalid. ⇒ Reenter in the "Section-name=address" format.
4	Invalid parameter input "xxx"	Command parameter specification is invalid. ⇒ Reenter the command parameter correctly.
5	Non relocatable file name	No relocatable filenames are input. Input relocatable filenames.
6	Internal error	Internal LINK77 error has occurred. ⇒ Contact the dealer where you purchased LINK77.

## A.1 List of Link Errors

Error No.	Error message	Meaning and actions
7	xxx relocatable format is mismatch	Relocatable format version of .R77 is different. ⇒ This error occurs when the version of the assembler or librarian differs from the linker's version. The .R77 and .LIB files to be linked must be created using the RASM77 and LIB77 programs of the same version as LINK77.
8	Program version is different	A different program version is declared by the pseudo instruction .VER. ⇒ Correct the version declaration by .VER so that the relocatable files to be linked are the same version; or, cancel version check with the "-V" parameter.
9	Unresolved label "xxx" in xxx	The indicated label or symbol is declared for external referencing but not defined in the indicated section. ⇒ Link the program in which the label or symbol is declared as a public label or symbol.
10	"xxx" is multiple defined in xxx. others in xxx	The indicated label or symbol is defined more than once. ⇒ Change the label or symbol name.
11	Location overlap. SECTION=xxx ADDRESS=xxx in xxx	Address space for the indicated section is overlapped. ⇒ Check the address assignments for the section, eliminate address overlapping. (If the section is an absolute type, the pseudo instruction .ORG in the source file must be modified.)
12	SECTION xxx is an absolute	Beginning address is specified for an absolute attribute section with a section control command. ⇒ Delete address specification from command input, or change the attribute of the section to relocatable.

## APPENDIX A. ERROR MESSAGES

Error No.	Error message	Meaning and actions
14	Can't find SECTION xxx	The indicated section cannot be found. ⇒ Correctly specify section information. (Note that uppercase and lowercase are recognized for section)
15	Can't create xxx	The indicated file cannot be created. ⇒ Check specification of the "-O" parameter and re-input.
16	File seek error xxx	Seek error has occurred on the indicated file. ⇒ This is an OS error. Usually, this error is caused by hardware malfunction of disk drive.
17	Expression value is out of range SECTION=xxx ADDRESS=xxx OFFSET=xxx	The result of operation at the indicated location exceeds the limit. (Error location is specified by section name, absolute address and offset from the beginning of section.) ⇒ Correct the program so that the limit will not be exceeded. This also includes the case where the target address cannot be accessed in the current address range.
18	Out of disk space	Disk space is insufficient for file output. ⇒ Make free space on the disk.
19	Relative jump out of range SECTION=xxx ADDRESS=xxx OFFSET=xxx	Relative jump address in the indicate location is out of accessing range. ⇒ Correct the program so that the jump destination label is within the accessing range. (Error location is specified by section name, absolute address and offset from the beginning of section.)

## A.1 List of Link Errors

Error No.	Error message	Meaning and actions
20	Expression is out of DP range. SECTION=xxx ADDRESS=xxx OFFSET=xxx	The result of expression processed in the direct addressing mode exceeds the range of DPR value declared by .DP through +0FFH. Correct the program so that the result will be in the range indicated above. (Error location is specified by section name, absolute address and offset from the beginning of section.)
21	Expression is out of DT range. SECTION=xxx ADDRESS=xxx OFFSET=xxx	The result of an expression processed in the absolute addressing mode exceeded the bank area of data bank register declared by .DT. Correct the program so that the result will be in the data bank register's bank range. (Error location is specified by section name, absolute address and offset from the beginning of section.)
22	Out of maximum program size	Program size exceeds 16M bytes (0FFFFFFFH). Reduce the size of the program.
23	Section type mismatch in SECTION xxx	ROM and RAM types are specified in the indicated section. Change the section to all ROM type or all RAM type.
24	Pointer length mismatch	The pointer length used by C compiler is not uniform. ⇒ Unify the value set with the pseudo instruction ".POINTER".



## A.2 Warning Message

### A.2 List of Warning Messages

<b>Warning No.</b>	<b>Warning message</b>	<b>Meaning and actions</b>
0	XXX instruction exist at end of bank (address XXXXH)	The instruction XXX (RTS, JMP, JSR) is on bank boundary (address XXXXH) (This warning also appears when there is data identical to machine code of instruction XXX.)
1	BRAL specified address is xxFFFFh (description address xxxh)	It is possible that BRAL mnemonic which jump address is xxFFFFh was located at xxxh address. Confirm that the found data is under BRAL mnemonic, because LINK77 searches machine language file for the machine code pattern.

# APPENDIX B

## MITSUBISHI Original HEX Format

### B.1 MITSUBISHI Original HEX Format

The MITSUBISHI original HEX format uses partially modified extended Intel HEX format address records to enable expression of the entire memory space of 7700 Family. Shown below is the address record format:

```
:02 0000 FF 00xx xx <RET>  
 1  2    3  4  5
```

The fields (1,2,3,4,5) specify the following information:

1. Specifies the delimiter (:) and data count (fixed to 02).
2. Specifies the address field (fixed to 0000).
3. Specifies the record type (fixed to FF).
4. High-order byte is fixed to 00. Low-order byte specifies the contents of subsequent data record address bits 16-23 (xx portion).
5. Specifies a 1-byte checksum.

Reference:

Shown below is the address record format of the extended Intel HEX format:

```
:02 0000 02 xxxx xx <RET>  
 1  2    3  4  5
```

The fields (1,2,3,4,5) specify the following information:

1. Specifies the delimiter (:) and data count (fixed to 02).
2. Specifies the address field (fixed to 0000).
3. Specifies the record type (fixed to 02).

## APPENDIX B. ORIGINAL HEX FORMAT FOR 7700 Family

---

4. Specifies 2-byte extended address. This address is shifted 4 bits and then added to the 2-byte address in the data record field to generate the final 20 bit address. The extended Intel HEX format record address output by LINK77 uses the high-order 4 bits to specify the page address and always leaves the low-order 12 bits 0.
5. Specifies a 1-byte checksum.

## B.2 HEXTOS2

### B.2.1 Overview

HEXTOS2 is a HEX file converter for the 7700 Family. It converts the Intel HEX format file created by LINK77 and 7700 Family custom HEX format machine language file (extension .HEX) into Motorola S format machine language file (extension .S2). The startup procedure is described below.

### B.2.2 Function

- The name of the HEX file is specified on the command line. The extension may be omitted. However, an error will occur if there is no file with the extension .HEX or there are more than one file with the same filename.
- The output file name is formed by appending the extension '.S2' to the input file name. The output file name cannot be specified.
- The output file consists of Motorola S format S2 data records and S9 end record.
- Following is an example of starting HEXTOS2.

```
A>HEXTOS2 sample.hex <RET>
MOTOROLA S2-RECORD GENERATION UTILITY V.2.00.00C
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved

Complete!!

A>
```

PART 4

LIBRARIAN FOR  
7700 FAMILY



**LIB77 OPERATION MANUAL**

---

# Table of Contents

## Chapter 1. Organization of LIB77 Operation Manual

## Chapter 2. Overview

2.1 Functions .....	2
2.2 Features .....	2
2.3 Files Created .....	3
2.4 Organization of LST File .....	4

## Chapter 3. Operation

3.1 Starting LIB77 .....	8
3.2 Input Parameters .....	8
3.2.1 Library Filename .....	8
3.2.2 Relocatable Filename .....	9
3.2.3 Command Parameters .....	9
3.2.4 Detailed Explanation of Command Parameters .....	10
3.3 Input Modes .....	12
3.3.1 Command Line Input Mode .....	12
3.3.2 Command File Input Mode .....	13
3.4 Errors .....	15
3.4.1 Error Types .....	15
3.4.2 Return Values to MS-DOS .....	16
3.5 Environment Variables .....	16

## Appendix A. Error Messages

A.1 System Error Messages .....	17
A.2 Librarian Error Messages .....	17

---

## List of Figures

Figure 2.1 LST File Output Example 1 (List of Module Names) .....	4
Figure 2.2 LST File Output Example 2 (List of Global Labels and Symbols) .....	5
Figure 2.3 LST File Output Example 3 (List of Global Labels and Symbols by Module) .....	7
Figure 3.1 Example of Command Input Line - 1 (Module Deletion) .....	13
Figure 3.2 Example of Command Input Line - 2 (Relocatable File Addition) .....	13
Figure 3.3 Specifying a Command File .....	14
Figure 3.4 Example of Command File Specification .....	14
Figure 3.5 Example of LIB77 Normal Termination Screen .....	14
Figure 3.6 Help Screen When There is a Command Line Error .....	15

---

## List of Table

Table 3.1 List of Command Parameters .....	9
Table 3.2 List of Error Levels .....	16
Table A.1 List of System Error Messages .....	18
Table A.2 Librarian Error Messages .....	19

---

# CHAPTER 1

## Organization of LIB77 Operation Manual

The LIB77 Operation Manual consists of the following chapters:

- Chapter 2. Overview  
Describes the basic functions of the LIB77 librarian and the files created by LIB77.
- Chapter 3. Operation  
Explains how to input LIB77 commands.
- Appendix A. Error Messages  
Lists the messages output by LIB77 along with explanation of the errors and the actions to be taken.



# CHAPTER 2

## Overview

The LIB77 librarian is a software product for managing the relocatable files created by RASM77 in the library format. By placing frequently used subroutines in a library, assembly time can be reduced and reuse of routines can be promoted.

### 2.1 Functions

LIB77 can be used along with LINK77<sup>1</sup>. To maximize the utility of these software products, LIB77 offers the following functions:

1. Creates and corrects library files that can be referenced by LINK77.
2. Catalogs relocatable files in a library file.
3. Deletes unnecessary relocatable files from a library file.
4. Updates old relocatable files that are cataloged in a library file with newly created relocatable files.
5. Restores relocatable files that have been cataloged in a library file to the precataloging condition.
6. Displays information on relocatable files that are cataloged in a library file.

### 2.2 Features

1. Speed-up linkage editing

By placing relocatable files in a library file, the necessary file information can be retrieved quickly during link, resulting in high-speed linkage processing.

2. When updating relocatable files that are cataloged in a library file, LIB77 compares the file modification dates and updates only the most recent version of each relocatable file. (When the command parameter "-U" is specified.)

---

<sup>1</sup> .LINK77 is the program name of 7700 Family linkage editor.

## 2.3 Files Created

LIB77 creates four types of files:

1. Library file
  - Library file created by editing relocatable files that have been created by RASM77 and adding the label and symbol index.
  - Within a library file, each relocatable file is managed as a module. (Hereafter, relocatable files in a library file are referred to as modules.)
  - Module name is the same as the relocatable filename, including the extension.
  - Library files are not in list format, so that they should not be printed or displayed on screen.
  - Library files have the extension, .LIB.
  
2. List file (hereafter referred to as LST file)
  - A LST file contains tables of the names of relocatable files, global labels and symbols, etc. in a library file.
  - A LST file is created when the command parameter “-L” is specified.
  - LST file has the extension, .LST.
  - Organization of LST file is described in the next section.
  
3. Relocatable files
  - The relocatable files created by LIB77 are reproduction of the relocatable files that have been cataloged in a library file.
  - Relocatable files are created when the command parameter “-X” is specified.
  - The extracted relocatable files are identical to the pre-cataloging relocatable files that have been created by RASM77.
  - Relocatable files have the extension, .R77.
  
4. Backup file
  - When a library file is updated, LIB77 retains the pre-updating library file as a backup file.
  - Backup file is always created when a library file is updated.
  - Backup files have the extension, .BAK.

### 2.4 Organization of LST File

Figures 2.1, 2.2 and 2.3 are sample LST file printouts. A LST file includes the following information:

1. Module names table (Figure 2.1)

The module names table contains the following information contained in a library file:

- **Module name:** Shows the module names that are cataloged in a library file. Module names are shown in the order in which they were cataloged in the library file.
- **Offset:** Shows the bytes count (in hexadecimal) from the beginning of the library file to the beginning of the module.
- **Module size:** Shows the memory size (in hexadecimal) of each module.

```
LIB77 librarian V.5.00.00                               date 1990-Aug-16 14:30 page 1

Library file name:      SAMPLE.LIB
Relocatable format:    VER.A
Last update time:      1990-Aug-16 15:30
Number of modules:     2
Number of global symbol: 10

Module_name:
getvalue ..... Offset: 00000000H Module size: 00000100H
gettoken ..... Offset: 00000180H Module size: 00000400H
```

**Figure 2.1 LST File Output Example 1 (List of Module Names)**

2. Global labels and symbols tables

Two tables are created:

- **PUBLIC symbol table**
  - Symbol\_name:** Shows a public label or symbol. A public symbol that has been declared by the pseudo instruction `.EQU` is followed by “e”.
  - Module\_name:** Lists the names of the modules that contain the public label or symbol.
- **EXTERN symbol table**
  - Symbol\_name:** Shows an external label or symbol name.
  - Module\_name:** Lists the names of the modules that externally reference the external label or symbol.

```
LIB77 librarian V.5.00.00                                date 1990-Aug-16 14:30 page 2
PUBLIC symbol table (symbol count = 0010)

Symbol_name      Module_name      Symbol_name      Module_name

_chgbin.....    getvalue          _chgdigit.....  getvalue
_chghex.....    getvalue          _chgoc.....     getvalue
_gettoken.....  gettoken          _getvalue.....  getvalue
_one(e).....    getvalue          _two(e).....    getvalue

LIB77 librarian V.5.00.00                                date 1990-Aug-16 14:30 page 3
EXTERN symbol table (symbol count = 0010)

Symbol_name      Module_name      Symbol_name      Module_name

_chgbil.....    getvalue          _chgdigi1.....  getvalue
_chghel.....    getvalue          _chgocl.....    getvalue
_gettokel.....  gettoken          _getvalul.....  getvalue
_isbil.....     getvalue          _ishel.....     getvalue
```

**Figure 2.2 LST File Output Example 2 (List of Global Labels and Symbols)**

## CHAPTER 2. OVERVIEW

---

### 3. Global labels and symbols tables by module

Two tables, PUBLIC symbol table and EXTERN symbol table, are created. In each of these tables, a module name is followed by a list of global labels and symbols in that module.

```
LIB77 librarian V.5.00.00                                date 1990-Aug-16 14:30 page 4

PUBLIC symbol table

Module_name : gettoken (symbol count = 0002)

_gettoken  __gettoken

Module_name : getvalue (symbol count = 0008)

_chgbin _chgdigit  _chghex _chgoc1
_gettoken _getvalue _one(e) _two(e)

LIB77 librarian V.5.00.00                                date 1990-Aug-16 14:30 page 5

EXTERN symbol table

Module_name : gettoken (symbol count = 0002)

_gettoken1 __gettoken1

Module_name : getvalue (symbol count = 0008)

_chgb11 _chgdig11  _chgh11 _chgoc1
_getval11 _isb11 _ish11  _isoc1
```

**Figure 2.3 LST File Output Example 3 (List of Global Labels and Symbols by Module)**

# CHAPTER 3

## Operation

### 3.1 Starting LIB77

To execute LIB77, the following information (input parameters) must be input:

1. Library filenames
2. Relocatable file names (required)
3. Command parameters

Input parameters for LIB77 execution can be input in either of the following two modes:

1. Command line input mode
2. Command file input mode

Same input parameters are used for the two input modes available. Also, same commands can be executed regardless of the input mode. The input parameters are explained in Section 3.2, and each input mode is explained in detail by referring to examples in Section 3.3.

### 3.2 Input Parameters

#### 3.2.1 Library Filename

1. Library filename must be specified.
2. Library filename to be edited is specified after one space following the command parameter “-O”.
3. Library filename may be specified with a directory path name. If directory path is not specified, the environment variable “LIB77” is referenced as the directory path.
4. Library filename extension, .LIB, may be omitted.

### 3.2.2 Relocatable Filename

1. More than one relocatable filename may be specified by using space as the delimiter.
2. Relocatable filenames to be processed are input after one space following the command parameter “-F”.
3. Filename may be specified with a directory path. If only filename is specified, LIB77 processes a file in the current drive’s current directory.
4. Relocatable filename extension, .R77, may be omitted.

### 3.2.3 Command Parameters

Command parameters are used to control library file manipulation, LIB77 output files, etc. Table 3.1 lists the command parameters available with LIB77.

**Table 3.1 List of Command Parameters**

No. <sup>1</sup>	Command parameter <sup>2</sup>	Description
1	-O	Specifies the name of library file to be edited.
2	-F	Specifies the names of relocatable files to be added to or deleted from a library file.
3	-A	Adds relocatable files to a library file.
	-R	Updates relocatable files in a library file.
	-D	Deletes specified relocatable files from a library file.
	-L	Outputs an LST file.
	-X	Extracts specified relocatable files from a library file and restores them to their pre-cataloging condition. The reproduced relocatable files are output to the current directory.
4	-V	Displays name of the file being processed on the screen.
	-U	Specifies to update only the most recent files when updating relocatable files in a library file.



Notes:

1. No. has the following meaning:
  - 1: Required item. Name of the library file to be edited must always be specified.
  - 2: When adding, updating, deleting or extracting relocatable files by specifying the command parameter “-A”, “-R”, “-D” or “-X”, the names of relocatable files to be processed must be specified.
  - 3: One and only one of these five command parameters must be specified.
  - 4: May be specified as needed.
2. No distinction is made between uppercase and lowercase characters in the command parameter. Therefore, “-A” and “-a” are both valid.
3. The relocatable files to be stored in the library should be assembled without specifying the assembly option “-S” or “-C”. File containing local symbol information and debug information slows library processing and significantly increases the library file size.

### 3.2.4 Detailed Explanation of Command Parameters

Command parameters are explained in detail below.

1. -O
  - Specifies the name of the library file to be edited.
  - Directory path name may be specified with library filename.

The following is an example of specifying file TEST.LIB in directory USR on drive A as the edit library file.

Example: A>LIB77 -LO B:\USR\TEST<RET>
  - If no directory path is specified, the MS-DOS environment variable “LIB77” is referenced as the directory path. If the environment variable is set as shown below, a file in the \USR directory of drive B is processed.

Example: A>SET LIB77=B:\USR
  - If directory path is omitted and environment variable has not been set, a file in the current drive’s current directory is processed.
  - If file extension is not specified, the default extension, .LIB, is used.
  - Files with extension other than .LIB can be processed by specifying full filenames.

2. -F
  - Specifies the names of relocatable files that are to be added to, updated, deleted from or extracted from a library file.
  - More than one filename may be specified by using space as the delimiter.
  - Directory path name may be specified with filename.  
The following is an example of specifying file SUB1.R77 in directory WORK on drive B and file SUB2.R77 on drive C as relocatable files.  
Example: A>LIBT7 -A0 TEST.LIB -F B:\WORK\SUB1C:SUB2 <RET>
  - If directory path specification is omitted, file in the current drive's current directory is processed.
  - If file extension is not specified, the default extension, .R77, is used.
  - Files with extension other than . R77 can be processed by specifying full filenames.
  
3. -A
  - Adds relocatable files to a library file.
  - The relocatable files specified by "-F" are appended at the end of the library file in the order in which they are specified.
  - If there are two files with identical contents, the duplicate label/symbol definition error will occur. If a relocatable file with the same name as that of a module already in the library file is specified, no error will result during cataloging; however, only the initially cataloged module will be processed when the file is specified in subsequent command specifying "-F".
  
4. -R
  - Updates modules (relocatable files) in a library file.
  - By using this command parameter in conjunction with "-U", only the files with more recent updating date than the modules in the library file can be updated.
  
5. -D
  - Deletes specified modules from a library file.
  
6. -L
  - Outputs a LST file.
  - When filenames are specified by "-F", information on only the specified relocatable files are output to a LST file.
  - When filenames are not specified by "-F", information on all modules in the library file are output to a LST file.
  - LST file has the extension, .LST.

### 7. -X

- Reproduces specified relocatable files by extracting them from a library file and restoring them in the pre-cataloging condition. The modification date of the relocatable files will be the LIB77 execution date
- The reproduced relocatable files will be identical to the original relocatable files prior to cataloging in the library file.
- The reproduced relocatable files are output to the current directory.
- When no relocatable filename is specified by “-F”, all modules in the library file are extracted and reproduced.
- Execution of this command parameter does not change the contents of the library file.
- Reproduced files have the extension, .R77.

### 8. -V

- Displays on the screen the file currently processed by LIB77.
- This command parameter must be specified along with “-A”, “-C”, “-D”, “-X”, or “-L”.

### 9. -U

- Updates only the most recent files when updating (“-R” specified) modules in a library file.
- For each module in the library file subject to updating, LIB77 compares the updating date of the relocatable file specified by “-F” to the updating date of the module as cataloged in the library file, and updates only if the relocatable file specified by “-F” is more recent.
- Information in OS directory is used as the updating date of relocatable file. Thus, this function should not be used on a computer that does not have the calendar function.

## 3.3 Input Modes

### 3.3.1 Command Line Input Mode

The command line input mode has the following features:

1. This input mode allows input of librarian commands from the OS command prompt.
2. Because OS limits command length, this input mode should be used when only few files and command parameters are needed.

3. This input mode can also be used when specifying execution commands in a batch file or a make file.
4. Figure 3.1 illustrates an example of deleting a module, FILE1.R77, from a library file, TEST.LIB.
5. Figure 3.2 illustrates an example of adding relocatable files, FILE1.R77 and FILE2.R77, to a library file, TEST.LIB.

```
A>LIB77 -D -O TEST.LIB -F FILE1<RET>
```

**Figure 3.1 Example of Command Input Line - 1 (Module Deletion)**

```
A>LIB77 -AO TEST.LIB -F FILE2 FILE3<RET>
```

**Figure 3.2 Example of Command Input Line - 2 (Relocatable File Addition)**

### 3.3.2 Command File Input Mode

The command file input mode has the following features:

1. In this input mode, the operation to be performed by LIB77 is stored in a command file using an editor in advance, and the name of this command file is specified when starting LIB77.
2. The command file input mode is convenient when the command line input mode cannot be used because there are too many file names and command characters.
3. The command file input mode can also be used when specifying execution commands in a batch file or a make file.
4. Command filename is entered using @ as prefix when starting LIB77 prompt as shown in Figure 3.3. In the example shown in Figure 3.3, LIB77 executes the contents of the file, CMD.DAT.
5. Specifications in a command file are same as when using the command line input mode (except that "LIB77" need not be specified to start LIB77). Line feed code is interpreted as a space so that a long command can be specified on multiple lines. The specifications in the example shown in Figure 3.2 can be created in a command file as illustrated in Figure 3.4.

```
A>LIB77 @CMD.DAT<RET>
```

**Figure 3.3 Specifying a Command File**

```
-AO  
TEST.LIB  
-F  
FILE2  
FILE3
```

**Figure 3.4 Example of Command File Specification**

When commands are input correctly, LIB77 starts processing. When LIB77 completes execution of all commands specified, it outputs a termination message on the screen and terminates. Figure 3.5 shows the screen display when LIB77 terminates normally.

```
A>LIB77 -AVO TEST.LIB -F SUB_1 SUB_100<RET>  
7700 Family LIBRARY MANAGER V.2.00.10  
Copyright 1998, MITSUBISHI ELECTRIC CORPORAION  
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION  
All right reserved.  
< test.lib > Create  
APPEND FILE = sub_1,sub_100  
MODULE COUNT      000002  
GLOBAL SYMBOL COUNT 000019  
A>
```

**Figure 3.5 Example of LIB77 Normal Termination Screen**

## 3.4 Errors

### 3.4.1 Error Types

The following types of errors may occur during execution of LIB77:

1. MS-DOS errors

Errors related to the MS-DOS environment in which LIB77 is executed. These errors include disk and memory shortages. When such an error occurs, the error message list in Appendix A should be checked and appropriate MS-DOS command should be entered.

2. LIB77 command input errors

These are the errors in LIB77 starting command input. When error is detected in command input, LIB77 outputs a HELP screen similar to that illustrated in Figure 3.6. Command input should be checked against the descriptions in this chapter, and a correct command line must be re-entered.

3. Errors in relocatable files to be processed

These are errors in the contents of the relocatable files being processed such as duplicate public label definitions. The files must be corrected by referring to the LST file.

```
A>LIB77<RET>
7700 Family LIBRARY MANAGER V.2.00.10
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All right reserved.

Usage: A>LIB77 -[ardxluv]o <filename> [-f <filename> ...]

-o : library file name
-f : relocatable file names
-a : append command
-r : replace command
-d : delete command
-x : extract command
-l : listout command
-u : update check (option)
-v : verbose (option)
```

**Figure 3.6 Help Screen When There is a Command Line Error**

When LIB77 detects an error, it outputs error information to the screen. The information should be checked against the error message list in Appendix A, and appropriate action must be taken.

### 3.4.2 Return Values to OS

When using an OS batch execution file, there are times when it is desirable to modify processing according to results of execution. LIB77 returns one of four error level return values to OS depending on the result of execution as summarized in Table3.2. For explanation of how to utilize these error level values, refer to an OS reference guide.

**Table 3.2 List of Error Levels**

Error level	Execution result
0	Normal termination.
1	Error in contents of library file or relocatable files being processed
2	LIB77 Command input error
3	OS error

## 3.5 Environment Variables

LIB77 uses the following environment variables:

#### 1. TMP77

This variable specifies the name of the directory in which temporary files are created when the librarian is executed. If this environment variable is not set, the temporary files are created in the current directory. The following is an example of how to set this environment variable:

Example:     A>SET TMP77=\USR\TMP<RET>

#### 2. LIB77

The environment variable "LIB77" can be used to specify the name of the library file to be edited. If the library file is not in the current directory, the directory path name can be omitted during LIB77 startup if it is set to the environment variable "LIB77". The following is an example of setting the environment variable to set the directory USR on drive B as the library file directory path.

Example:     A> SET LIB77=B:\USR<RET>

# APPENDIX A

## Error Messages

### A.1 System Error Messages

When LIB77 detects a system error during execution, it outputs an error message on the screen and cancels processing. Table A.1 lists the system error messages output by LIB77.

### A.2 Librarian Error Messages

When LIB77 detects a processing error during execution, it outputs an error message on the screen and cancels processing. Table A.2 lists the librarian error messages output by LIB77.



## APPENDIX A. ERROR MESSAGES

---

Table A.1 List of System Error Messages

Error message	Meaning and actions
Usage A>LIB77 - [adluvzx] o <filename> [-f<filename>...]	Command input is invalid. ⇒ Refer to the HELP screen, and re-input the command correctly.
Can' t open xxx	The indicated file cannot be found. ⇒ Check if the files specified with the command parameter "-O" or "-F" are in the specified directories.
Can' t create xxx	The indicated file cannot be created. ⇒ Check specification of "-O" command parameter, and re-input.
Out of disk space <sup>1</sup>	Disk space is insufficient to output file. ⇒ Make free space on the disk.
Input file read error xxx	An error has occurred while reading an input file. ⇒ This is an OS error. Usually, this error is caused by a hardware malfunction of disk drive.
Internal error	An internal LIB77 error has occurred. ⇒ Contact the dealer where you purchased LIB77.
File seek error xxx	Seek error has occurred on the indicated file. ⇒ This is an OS error. Usually, this error is caused by hardware malfunction of disk drive.

Note:

1. Free disk space equal to 2-3 times the library file size is necessary in order to execute LIB77 because intermediate work files are created during execution.

**Table A.2 Librarian Error Messages**

<b>Error message</b>	<b>Meaning and action</b>
xxx is a multiple defined in xxx. others in xxx	Public label or symbol is defined more than once. ⇒ Check the public label or symbol definition in the LST file.
xxx module is not in the library	The indicated module cannot be found in the library file. ⇒ Check the module name in the LST file.
Invalid module or library	The specified relocatable file and corresponding module in the library file are in different formats. ⇒ The library file and relocatable file to be edited must be created with RASM77 of the same version.
xxx command file not found	The indicated command file cannot be found. ⇒ Check the command file specified.
Out of heap space	Memory space is insufficient to execute the librarian program. ⇒ Reduce the number of global labels.
Too many object modules	There are too many modules in the library file. ⇒ Split the library file. One library file can contain only up to 500 modules.
CPU number error	The specified library file or relocatable file was not created by RASM77. ⇒ Check the specified library file, or relocatable file.

PART 5

CROSS REFERENCER FOR  
7700 FAMILY



**CRF77 OPERATION MANUAL**

---

# Table of Contents

## Chapter 1. Organization of CRF77 Operation Manual

## Chapter 2. Overview

2.1 Functions .....	2
2.2 Files Created .....	2
2.3 Organization of CRF File .....	2

## Chapter 3. Operation

3.1 Starting CRF77 .....	4
3.2 Input Parameters .....	4
3.2.1 Source Filename .....	4
3.2.2 Command Parameters .....	4
3.3 Input Mode .....	4
3.3.1 Command Line Input Mode .....	4
3.4 Errors .....	5
3.4.1 Error Types .....	5
3.4.2 Return Values to MDOS .....	5
3.5 Environment Variables .....	5

## Appendix A. Error Messages

A.1 System Error Messages .....	8
A.2 Cross-reference Error Messages .....	9

---

## List of Figures

Figure 2.1 CRF File Example .....	3
Figure 3.1 Command Input Line Example .....	6
Figure 3.2 Help Screen when a Command Line Error Occurs .....	6
Figure 3.3 Example of Error Display .....	7

---

## List of Tables

Table 3.1 List of Command Parameters .....	6
Table 3.2 List of Error Levels .....	7
Table A.1 List of System Error Messages .....	8
Table A.2 Cross-reference Error Message .....	9

---

# CHAPTER 1

## Organization of CRF77 Operation Manual

The CRF77 Operation Manual consists of the following chapters:

- Chapter 2. Overview  
Describes the basic functions of the CRF77 and the files created by CRF77.
- Chapter 3. Operation  
Explains how to input CRF77 commands.
- Appendix A. Error Messages  
Lists the messages output by CRF77 along with explanation of the errors and the actions to be taken.

# CHAPTER 2

## Overview

CRF77 creates a cross-reference list of labels and symbols in source files. This listing makes it easy to see the relations between various sections of a program when debugging.

### 2.1 Functions

CRF77 facilitates the understanding of source files with the following functions:

1. Shows the type of label referencing instruction in the reference line number column.
2. Can include files with the pseudo instruction `. INCLUDE`.
3. Can output a header line by the pseudo instruction `. PAGE`.

### 2.2 Files Created

CRF77 creates the following file:

1. Cross-reference file (hereafter referred to as CRF file)
  - Contains a cross-reference listing of label and symbol names.
  - There are 80 columns per line (fixed), and 57 lines per page (fixed).
  - This file can be printed out for use in debugging and editing.
  - Cross-reference file has the extension, `.CRF`.

### 2.3 Organization of CRF File

Figure 2.1 is a sample CRF file printouts. A CRF file includes the following information:



1. Label and symbol names, with line numbers where they are referenced or defined. Definition line number is suffix by a number sign (#), and subroutine referencing line is suffix by an ampersand (&).
2. Up to 32 characters are printed for each label or symbol name. The list is formatted for the longest name.
3. The title specified by the pseudo instruction. PAGE is printed as the list header. (Up to 30 characters)
4. CRF77 does not evaluate the values of labels and symbols in the source program. Accordingly, CRF77 cannot perform conditional assembly.

7700 Family CROSS REFERENCE V.2.10.10						P. 001
AO	3926	4285	8549	9079	9100	
AA	3884	5545	5668			
ABEND	9396&	9465&	9587#			
ABEND10	9588	9593#				
ABENDRT	9590#	9605				
ACCHK	1201#	1408				
ACCHK5	1213	1237#				
ACCHKE	1235	1239	1241	1249#		
ADDING	4994	5006#				
ADDING0	5013	5014	5016#			
ADDING1	5015	5019#				
ADDRESS	300	318	1025			
ADR_CHK	9154#					
ADR_OUT	8302	8336	9145#			
ADR_PNT	8157#					
ADR_PNT2	8149#					

**Figure 2.1 CRF File Example**

# CHAPTER 3

## Operation

### 3.1 Starting CRF77

To execute CRF77, the following information (input parameters) must be input:

1. Source filenames (required)
2. Command parameters

### 3.2 Input Parameters

#### 3.2.1 Source Filename

1. Source filename must always be specified.
2. If specification of file extension (.A77) is omitted, the default extension, .A77, is used.
3. A file whose extension is not .A77 (e.g., .ASM) can be processed by specifying full filename.
4. Drive name may be specified with filename. If only the filename is specified, a file in the current drive is processed. Directory path cannot be specified.
5. Up to 16 source filenames may be specified.

#### 3.2.2 Command Parameters

Command parameters are used to specify whether or not to detect the pseudo instruction. INCLUDE in the source files and to specify the drive name for the output file.

Table 3.1 List of Command Parameters

Command parameter	Description
-O	Specifies the drive name and directory path to which the cross reference file is output. The coding format is as follows: Example: A> CRF77 SRCFILE -OC:\TMP<RET> Outputs the cross reference file to directory TMP on drive C.
-I	Specifies to ignore the pseudo instruction .INCLUDE. The .INCLUDE pseudo instruction for RASM77 allows nesting up to 9 levels, but the .INCLUDE pseudo instruction for CRF77 does not allow nesting. Therefore, use this command option '-I' when processing with CRF77 source containing .INCLUDE pseudo instruction nests.

## 3.3 Input Mode

### 3.3.1 Command Line Input Mode

CRF77 is started when a command line is input from the command prompt. Figure 3.1 shows how to start CRF77.

```
A>CRF77 SRCFILE1 SRCFILE2 SRCFILE3<RET>
```

Figure 3.1 Command Input Line Example

When CRF77 detects an error during command line input, it outputs the HELP screen and cancels processing (see Figure 3.2).

```
A>CRF77<RET>
7700 Family CROSS REFERENCE V.2.10.10
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Usage: crf77 <filename> [-ifilename,..] [-opath]

    -i : not include specified files ( use -ifilename,... )
    -o : select drive and directory for output ( use -otmp )
```

Figure 3.2 Help Screen when a Command Line Error Occurs

### 3.4 Errors

#### 3.4.1 Error Types

The following types of errors may occur during execution of CRF77:

1. OS errors

Errors related to the environment in which CRF77 is executed. These errors include disk and memory shortages. When such an error occurs, the error message list in Appendix A should be checked and appropriate OS command should be entered.

2. CRF77 command line input errors

These are the errors in CRF77 startup command input. Command input should be checked against the descriptions in this chapter, and a correct command line must be re-input.

3. Source file error

This error occurs when the source file specified by the pseudo instruction . INCLUDE cannot be found.

When CRF77 detects an error, it outputs error information to the screen in the format shown in Figure 3.3. The information should be checked against the error message listing in Appendix A, and appropriate action must be taken.

```
A>CRF77 SRCFILE1<RET>
7700 Family CROSS REFERENCE V.2.10.10
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

now processing pass 1
now making cross reference ( SRCFILE1.A77 )
----+
  Out of heap space

A>
```

**Figure 3.3 Example of Error Display**

### 3.4.2 Return Values to OS

When using an OS batch execution file, there are times when it is desirable to modify processing according to results of execution. CRF77 returns one of four error levels to OS depending on the result of execution as summarized in Table 3.2. For explanation of how to utilize these error levels, refer to an OS reference guide.

**Table 3.2 List of Error Levels**

<b>Error level</b>	<b>Execution result</b>
0	Normal termination
1	Source file specified by . INCLUDE is missing
2	CRF77 command input error
3	OS error

## 3.5 Environment Variables

CRF77 does not use environment variables.

# APPENDIX A

## Error Messages

### A.1 System Error Messages

When CRF77 detects a system error during execution, it outputs an error message on the screen and cancels processing. Table A.1 lists the system error messages output by CRF77.

**Table A.1 List of System Error Messages**

<b>Error message</b>	<b>Meaning and actions</b>
Usage: A>crf77 <filename> [-d] -l[[filename,..]]	Command input is invalid. ⇒ Refer to the HELP screen, and re-input the command correctly.
Can't open xxx	The indicated file cannot be found. ⇒ Check the source filename, re-input correctly.
Can't create xxx	The indicated file cannot be created. ⇒ Make free space on the disk.
Out of disk space	Disk space is insufficient to output file. ⇒ Make free space on the disk.
Out of heap space	Memory space is insufficient to execute the cross-referencer. ⇒ Reduce the number of symbols or labels.

## **A.2 Cross-reference Error Messages**

When CRF77 detects an error during creation of a cross-reference listing, it outputs an error message on the screen and continues processing. Table A.2 lists the cross-reference error message output by CRF77.

**Table A.2 Cross-reference Error Message**

<b>Error message</b>	<b>Meaning and action</b>
Can't open include file xxx	Source file specified by .INCLUDE cannot be found. ⇒ Check directory contents.

# MEMO



# RASM77 V.5.10 User's Manual

---

Rev. 1.00  
August 01, 2003  
REJ10J0158-0100Z

COPYRIGHT ©2003 RENESAS TECHNOLOGY CORPORATION  
AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED

**RASM77 V.5.10**  
**User's Manual**



**Renesas Electronics Corporation**

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

REJ10J0158-0100Z