

**$\mu$ COM-17K Family  
4-Bit CMOS Microcomputer  
Development Tools**

**User's Manual**



# Contents

Page

## AS 17K Assembler User's Manual

Introduction.....	1-1- 1
<b>Part I Languages</b>	
<b>1. An outline.....</b>	<b>1-1- 1</b>
1.1 An outline of the assembler.....	1-1- 1
1.1.1 What is an assembler.....	1-1- 1
1.1.2 What is an absolute assembler.....	1-1- 1
1.1.3 What is a relocatable assembler.....	1-1- 2
1.1.4 The system development sequence for the $\mu$ PD17000.....	1-1- 3
1.1.5 A comparison of assemblers.....	1-1- 4
1.2 An overview of the functions of the $\mu$ PD17000.....	1-1- 6
1.2.1 Generating sequence files.....	1-1- 6
1.2.2 Generating source module files.....	1-1- 6
1.2.3 Assembling.....	1-1- 8
1.3 Before beginning program development.....	1-1-10
1.3.1 Restrictions on symbols.....	1-1-10
1.3.2 Restrictions on directives.....	1-1-11
1.3.3 Similar reserved words.....	1-1-12
1.3.5 Setting the time and date of the host.....	1-1-15
1.3.6 Restrictions on the number of source modules.....	1-1-15
1.4 Features of the assembler.....	1-1-16
1.4.1 PC-DOS assemblers.....	1-1-16
1.4.2 Capacities for program modularisation.....	1-1-16
1.4.3 Convenient built-in macro instructions.....	1-1-16
1.4.5 The documentation generation function.....	1-1-17
1.4.6 Two types of cross reference function.....	1-1-17
1.4.7 The assemble report function.....	1-1-18
1.4.8 The automatic object load function.....	1-1-18
1.4.9 Source program hierarchy creation.....	1-1-18
<b>2. Methods of entering source programs.....</b>	<b>1-2- 1</b>
2.1 The basic operations of source programs.....	1-2- 1
2.2 The structure of statements.....	1-2- 2
2.3 The tabulation function.....	1-2- 4
2.4 The character set.....	1-2- 5
2.4.1 Alphanumeric characters.....	1-2- 5
2.4.2 Numeric characters.....	1-2- 5
2.4.3 The use of special characters.....	1-2- 6

	Page
2.5 The symbol column .....	1-2- 8
2.5.1 Symbol types .....	1-2- 9
2.5.2 Rules for entering symbols .....	1-2- 9
2.6 The mnemonic column .....	1-2-12
2.7 The operand column .....	1-2-13
2.7.1 Entry format for the operand field .....	1-2-13
2.8 The comment column .....	1-2-17
2.9 Expressions and operators .....	1-2-18
2.9.1 Expressions .....	1-2-18
2.9.2 An outline of operators .....	1-2-22
2.9.3 Arithmetic operators .....	1-2-23
2.9.4 Logical operators .....	1-2-28
2.9.5 Comparative operators .....	1-2-32
2.9.6 Shift operators .....	1-2-38
2.9.7 Other operators .....	1-2-42
2.10 Functions .....	1-2-43
2.10.1 Type conversion functions .....	1-2-43
2.10.2 Location counter functions .....	1-2-46
2.11 Variables used when assembling .....	1-2-47
2.11.1 ZZZn .....	1-2-47
2.11.2 ZZZSKIP .....	1-2-49
<b>3. Directives and control instruction .....</b>	<b>1-3- 1</b>
3.1 An outline of virtual instructions and control instructions .....	1-3- 1
3.2 Virtual instructions .....	1-3- 2
3.2.1 The location counter control directive .....	1-3- 2
ORG	
3.2.2 Symbol definition directives .....	1-3- 4
DAT	
MEM	
FLG	
LAB	
SET	
3.2.3 Public definition and reference directives .....	1-3-17
PUBLIC, BELOW-ENDP	
EXTRN	
3.2.4 Data definition directives .....	1-3-22
DW, DB	

	Page
3.2.5 Assemble directives with conditions .....	1-3-26
IF, ELSE, ENDIF	
CASE, EXIT, OTHER, ENDCASE	
3.2.6 Iteration directives .....	1-3-32
REPT, ENDR	
IRP, ENDP	
EXITR	
3.2.7 The macro definition directive .....	1-3-39
MACRO-ENDM	
3.2.8 The symbol global declaration directive in macros .....	1-3-41
GLOBAL	
3.2.9 The assemble terminate directive .....	1-3-43
END	
3.3 Control instructions.....	1-3-45
3.3.1 Output list control instructions .....	1-3-46
TITLE	
EJECT	
LIST	
NOLIST	
SFCOND	
IFCOND	
C14344	
C4444	
3.3.2 Macro development print control instructions .....	1-3-60
SMAC	
NOMAC	
OMAC	
LMAC	
3.3.3 Source input control instructions .....	1-3-66
INCLUDE	
EOF	
3.3.4 Document generation control instructions .....	1-3-71
SUMMARY	
;, (TAG)	
3.4 Macro functions .....	1-3-75
3.4.1 Macro definition .....	1-3-75
3.4.2 Macro reference .....	1-3-76
3.4.3 Macro expansion .....	1-3-78
3.4.4 Examples of the use of macros .....	1-3-79

	Page
3.5 Document generation functions .....	1-3-85
3.5.1 Program summary .....	1-3-86
3.5.2 Module summary .....	1-3-87
3.5.3 Routine summary .....	1-3-87
3.5.4 Examples .....	1-3-90
3.5.5 Table of contents generation function .....	1-3-91
<b>4. Built-in macro directives .....</b>	<b>1-4- 1</b>
4.1 An overview of the built-in macro directives .....	1-4- 1
4.2 Built-in macro directives .....	1-4- 2
SKTn, SKFn	
SETn, CLRn	
NOTn	
INITFLG	
BANK	

## Part II Operations

<b>1. An outline of the product .....</b>	<b>1-5- 1</b>
1.1 Details of the product .....	1-5- 1
1.2 System configuration .....	1-5- 1
<b>2. Before executing .....</b>	<b>1-6- 1</b>
2.1 Making backup files .....	1-6- 1
2.2 Introduction to the sample program .....	1-6- 3
2.3 Procedures for assembling the sample programs .....	1-6- 4
<b>3. Sequence files .....</b>	<b>1-7- 1</b>
3.1 An outline .....	1-7- 1
3.2 Sequence file entry formats .....	1-7- 2
3.2.1 Total entry formats .....	1-7- 2
3.2.2 Device file name entry formats .....	1-7- 3
3.2.3 Assemble option entry formats .....	1-7- 4
3.2.4 Source file name entry formats .....	1-7- 5
3.3 Sequence file generation methods .....	1-7- 6

	Page
<b>4. Assembler functions</b> .....	1 -8- 1
4.1 Outline .....	1 -8- 1
4.2 Assembler input/output files .....	1 -8- 2
4.3 Assembler functions .....	1 -8- 5
4.3.1 The interim object module file output function .....	1 -8- 5
4.3.2 The link function .....	1 -8- 5
4.3.3 HEX file and PROM file output functions .....	1 -8- 5
4.3.4 Functions which reduce the assemble time .....	1 -8- 6
4.3.5 The assemble list output function .....	1 -8- 8
4.3.6 The cross-reference list file output function .....	1 -8- 8
4.3.7 The document file output function .....	1 -8- 8
4.3.8 The memory map file output function .....	1 -8- 8
4.3.9 The report file output function .....	1 -8- 9
4.4 Methods of starting up the assembler .....	1 -8-11
4.4.1 Files which must be input when starting up the assembler .....	1 -8-11
4.4.2 Convenient input files .....	1 -8-11
4.4.3 Methods of booting up the assembler .....	1 -8-12
4.4.4 Halting during assembly .....	1 -8-19
4.5 Assemble options .....	1- 8-20
4.5.1 The object file output control option .....	1- 8-23
4.5.2 The assemble list file output control option .....	1- 8-25
4.5.3 The cross-reference list file output control option .....	1- 8-27
4.5.4 The error skip control option .....	1- 8-29
4.5.5 The list output page row number control option .....	1- 8-30
4.5.6 The macro iteration directive development control option .....	1- 8-31
4.5.7 The list output column number control option .....	1- 8-32
4.5.8 The automatic load control option .....	1- 8-33
4.5.9 The conditional statement output control option .....	1- 8-34
4.5.10 The optional data output control option .....	1- 8-35
4.5.11 The data memory map file output control option .....	1- 8-36
4.5.12 The document file output control option .....	1- 8-38
4.5.13 The report file output control option .....	1- 8-40
4.5.14 The buried cross reference output control option .....	1- 8-42
4.5.15 The program name output control option .....	1- 8-44
4.5.16 The tab control option .....	1- 8-45
4.5.17 The form feed control option .....	1- 8-46
4.5.18 The assemble variable control option .....	1- 8-47
4.5.19 The public cross reference list file output control option .....	1- 8-48
4.5.20 The operational drive control option .....	1- 8-50
4.5.21 The program summary output option .....	1- 8-51
4.5.22 The SIMPLEHOST data output control .....	1- 8-52

	Page
5. Assembler output lists .....	1- 9- 1
5.1 Types of output lists .....	1- 9- 1
5.2 List output format controls .....	1- 9- 2
5.3 Outputting the header .....	1- 9- 3
5.4 Option data lists .....	1- 9- 4
5.5 Assemble lists .....	1- 9- 6
5.6 Cross reference lists .....	1- 9- 9
5.7 Memory maps .....	1- 9-12
5.7.1 Memory maps .....	1- 9-13
5.7.2 Flag maps .....	1- 9-15
5.7.3 Symbol maps .....	1- 9-16
5.8 Assemble reports .....	1- 9-18
5.8.1 Module reports .....	1- 9-18
5.8.2 Final phase reports .....	1- 9-21
5.8.3 Total reports .....	1- 9-22
5.9 Public cross-reference lists .....	1- 9-23
5.10 Documents .....	1- 9-25
5.10.1 Tables of contents .....	1- 9-25
5.10.2 The text of a document .....	1- 9-26
6. Error and warning messages .....	1-10- 1
6.1 Assembling errors .....	1-10- 1
6.2 Errors which relate to source programs .....	1-10- 3
Appendix 1	
Error messages when the program memory overflows .....	1-A-1
Appendix 2	
SIMPLEHOST .....	1-A-3



**IE-17K User's Manual**

<b>1. General information</b> .....	2-1- 1
1.1 Overview .....	2-1- 1
1.2 Characteristics of IE-17K .....	2-1- 2
1.2.1 Interface with target system .....	2-1- 2
1.2.2 Program memory .....	2-1- 2
1.2.3 How to emulate .....	2-1- 2
1.2.4 Break function .....	2-1- 2
1.2.5 Real time trace function .....	2-1- 3
1.2.6 Data memory coverage function .....	2-1- 3
1.2.7 Program memory coverage function .....	2-1- 4
1.2.8 Programmable pattern generator function .....	2-1- 4
1.2.9 Other characteristics .....	2-1- 4
1.3 Configuration .....	2-1- 5
1.3.1 System configuration diagram .....	2-1- 5
1.3.2 Block diagram .....	2-1- 6
<b>2. Specification</b> .....	2-2- 1
2.1 Main LSI .....	2-2- 1
2.2 Console interface .....	2-2- 1
2.3 Environment .....	2-2- 1
2.4 Power supply .....	2-2- 1
2.5 Built-in power supply .....	2-2- 1
2.6 Power consumption for each board .....	2-2- 2
2.7 External dimension (excluding projection) .....	2-2- 2
2.9 Accessories .....	2-2- 3
<b>3. Installation</b> .....	2-3- 1
3.1 Removing memory board supervisor .....	2-3- 1
3.2 Setting switches .....	2-3- 2
3.2.1 Setting the switches on memory board .....	2-3- 2
3.2.2 Setting switches on supervisor board .....	2-3- 4
3.3 Connection of connector .....	2-3- 6
3.3.1 Internal connector on memory board .....	2-3- 6
3.3.2 Internal connectors on supervisor board .....	2-3- 7
3.4 Installing SE board .....	2-3- 8
3.5 Connecting to host machine .....	2-3- 9
3.6 Connecting to PROM programmer .....	2-3-10
3.7 Connecting with target system .....	2-3-11

	Page
<b>4. Activation</b> .....	2-4- 1
4.1 Program loading .....	2-4- 1
<b>5. Commands</b> .....	2-5- 1
5.1 Command notation .....	2-5- 1
5.1.1 Command input form .....	2-5- 1
5.1.2 Form of command expression .....	2-5- 1
5.2 Prompt .....	2-5- 3
5.3 Commands .....	2-5- 4
5.3.1 Program memory control command .....	2-5- 4
(1) Initialize program memory (.IP) .....	2-5- 4
(2) Change program memory (.CP) .....	2-5- 4
(3) Dump program memory (.DP) .....	2-5- 4
(4) Find program memory (.FP) .....	2-5- 4
(5) Save program memory (.SP) .....	2-5- 4
(6) Load program memory (.LP) .....	2-5- 4
(7) Verification of program memory (.VP) .....	2-5- 4
(8) Output of PROM data (.XS) .....	2-5- 4
5.3.2 Control command for data memory .....	2-5-14
(1) Initialization of data memory (.ID) .....	2-5-14
(2) Change of data memory (.CD) .....	2-5-14
(3) Dump of data memory (.CD) .....	2-5-14
(4) Dump of all data memory (.D) .....	2-5-14
5.3.3 Emulation Command .....	2-5-20
(1) Reset (.R) .....	2-5-20
(2) Program run (.RN) .....	2-5-20
(3) Program run (Reset condition) (.BG) .....	2-5-20
(4) Break (.BK) .....	2-5-20
(5) Change start address of program (.CA) .....	2-5-20
(6) Step operation (.S) .....	2-5-20
(7) Display (.DS) .....	2-5-20
5.3.4 Break/trace condition control commands .....	2-5-28
(1) Change break/trace condition (.CC) .....	2-5-28
(2) Change trace ON/OFF condition (.CT) .....	2-5-28
(3) Dump break/trace condition (.DC) .....	2-5-28
(4) Dump trace table (.DT) .....	2-5-28
(5) Save break/trace condition (.SC) .....	2-5-28
(6) Load break/trace condition (.LC) .....	2-5-28
(7) Verify break/trace condition (.VC) .....	2-5-28

	Page
5.3.5 Coverage display command .....	2-5-54
(1) Dump coverage memory (.DM) .....	2-5-54
5.3.6 Program pattern generator (PPG) control commands .....	2-5-57
(1) Initialize PPG data (.IG) .....	2-5-57
(2) Change PPG data (.CG) .....	2-5-57
(3) Dump PPG data (.DG) .....	2-5-57
(4) Execute/stop PPG, set PPG operation mode (.EG) .....	2-5-57
(5) Save PPG data (.SG) .....	2-5-57
(6) Load PPG data (.LG) .....	2-5-57
(7) Verify PPG data (.VG) .....	2-5-57
5.3.7 Help command .....	2-5-67
(1) Lists all commands (.H) .....	2-5-67
5.3.8 Other commands .....	2-5-69
(1) Define macro (U) .....	2-5-69
(2) Execute macro (M) .....	2-5-69
(3) Dump macro (=C) .....	2-5-69
(4) Loop (< >) .....	2-5-69
<b>6. Programmable pulse generator .....</b>	<b>2-6- 1</b>
6.1 Displaying, modifying PPG data .....	2-6- 1
6.2 Setting the step rate .....	2-6- 2
6.3 Executing PPG, stopping PPG .....	2-6- 3
6.4 Notes on using the PPG .....	2-6- 4
6.5 PPG application example .....	2-6- 5
<b>7. Program execution .....</b>	<b>2-7- 1</b>
7.1 Real-time emulation .....	2-7- 1
7.2 Setting break points .....	2-7- 2
7.3 Single-step emulation .....	2-7- 3
<b>8. Programming the PROM for the SE board .....</b>	<b>2-8- 1</b>
<b>9. Error Messages .....</b>	<b>2-9- 1</b>
9.1 Error messages related to commands .....	2-9- 1
9.2 Hardware error .....	2-9- 3



# **AS 17K Assembler**

## **User's Manual**



### INTRODUCTION

1. The AS17K assembler supports all products in the 4-bit uPD17000 microcomputer series. As the products in the uPD1700 series are slightly different, it is necessary to have a device file corresponding to each one. Please make sure you purchase a device file with your assembler.
2. The AS17K assembler operates in the following environments:

Host computer	OS	Memory size
IBM PC-AT <sup>TM</sup>	PC-DOS <sup>TM</sup> version 3.1	512K bytes or more

The CONFIG.SYS file settings in PC-DOS should be as follows:

- Files equals 15 (within the range 15 to 20)
- Buffers equals 10 (10 or more)

IBM PC-AT<sup>TM</sup>, PC-DOS<sup>TM</sup> are trademarks of the IBM Corporation.

### 3. Status When Supplied

#### 3.1 The assembler

- (1) File name  
AS17K.EXE

- (2) Floppy disk formats

The assembler is supplied on 5-inch double-sided double density floppy discs.

#### 3.2 Device files

- (1) File name

D17XXX.DEV (optional)

- (2) Floppy disc formats

The device files are supplied on 5-inch double-sided double density floppy discs.

### 4. The meaning of the symbols used in this manual

... The same format continued

[ ] Whatever is between the square brackets can be abbreviated

{ } Select one of what is between the brackets

△ An en space or a tab

" " Characters between double quotation marks

CR Carriage return

LF Line feed

TAB Horizontal tab

ooo Represents a desired character string

xxx ditto

□□□ ditto

≡ Expresses a correspondence

< > Expresses a correspondence to what is in the parentheses.



### 5. File name restrictions

[Drive name:] [%directory name ...] file name    [.extension]
---

Drive name: The name of the drive in which the floppy disk containing files is set. If the drive name is omitted, the current drive will be selected.

File name: A character string of four or less em characters or eight en characters.

Extension: A character string of three or less en characters.



**PART I Language**

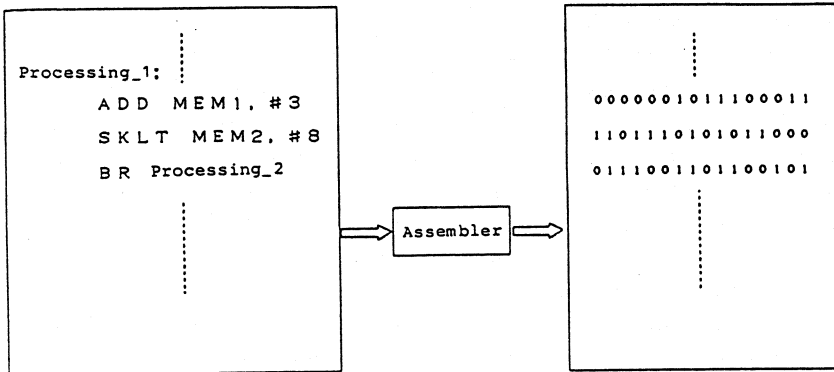


## CHAPTER 1 OUTLINE

### 1.1 Outline of the Assembler

#### 1.1.1 What is an assembler?

Machine language, and consists of zeros and ones. However, for humans, machine languages are extremely complex and difficult to learn. If we handle machine language by introducing a symbolic or assembly language, however, it is possible to enter programs which are much easier to understand. An assembler is a program which converts machine language, the only thing which a microcomputer can understand, into a symbolic language which is easy for humans to deal with.



Assembly language  
(Symbolic languages)

Machine language

Assemblers may be classified as absolute assemblers or relocatable assemblers. The AS17K is an absolute assembler, but it is different from conventional absolute assemblers, and permits split programming. Accordingly, while it is an absolute assembler, it possesses much of the character of a relocatable assembler also.

#### 1.1.2 What is an absolute assembler?

Machine language consists of instructions and data. Instructions indicate to the computer the type of action to be taken, while data are the values which are operated on at that time.

Data consists of constants and variables which are processed by operation instructions.

An absolute assembler is an assembler which determines absolutely the address allotted to instructions and data when converting to machine language. Thus, all addresses and data must be determined when assembling. This data is communicated to the assembler through a location counter control directive called ORG.

The machine language which is generated by an absolute assembler is stored in memory as is, and can be executed by the micro-computer. The so created machine language is known as an absolute object module. The element in the symbolic language which is its source is known as a source module.

### 1.1.3 What is a relocatable assembler?

The absolute object module created by an absolute assembler defines data and addresses absolutely. What is known as a relocatable assembler, in contrast to this, is an assembler which generates object modules which may be relocated at a desired address in memory. The machine language generated by the use of a relocatable assembler is known as a relocatable object module. The machine language contained in a relocatable object module cannot, as it is, be executed by a microcomputer as a program. This is because the addresses and the data have relative (temporary) values. A linker must be used to change the relocatable object module into a form in which it can be executed by a micro-computer.

### What is a linker?

A linker determines the positions of multiple relocatable object modules generated by a relocatable assembler, and the address reference relationships, and organizes them into a unity. The addresses and data which had been given relative values are allocated absolute values.

A single arrangement of modules output from the linker is known as a load module. The load module cannot, as it is, be caused to execute by the microprocessor. It is necessary for it to be converted into a form in which it can be executed by the micro-processor.

### 1.1.4 The system development sequence for uPD17000

Figure 1.1 shows the total systems development sequence used for the uPD17000 series. A detailed flow chart of the development of the software, also, is shown in Figure 1.2.

Figure 1.1 System Development Sequence

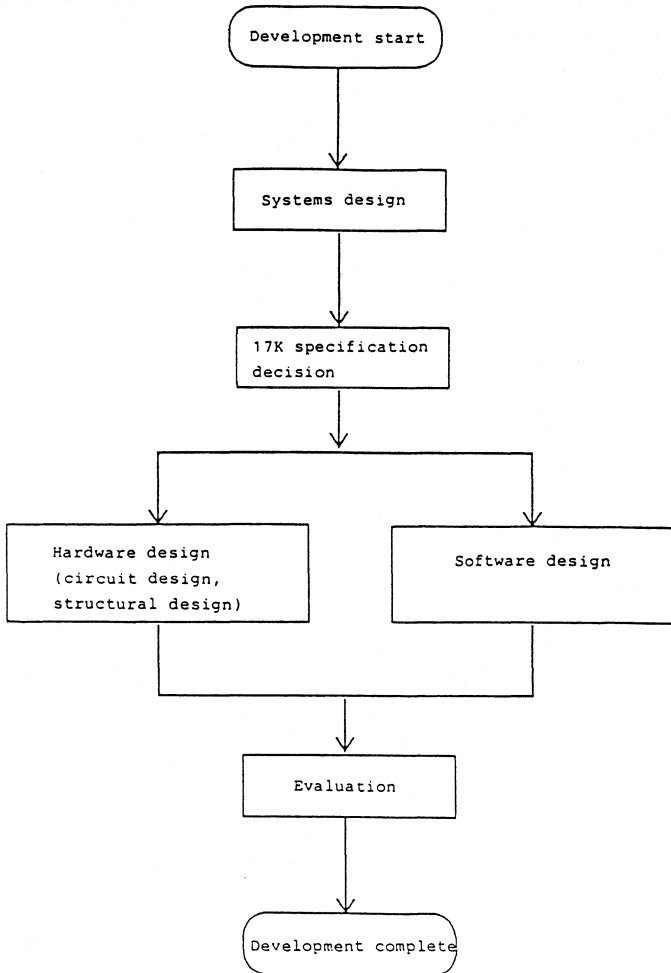
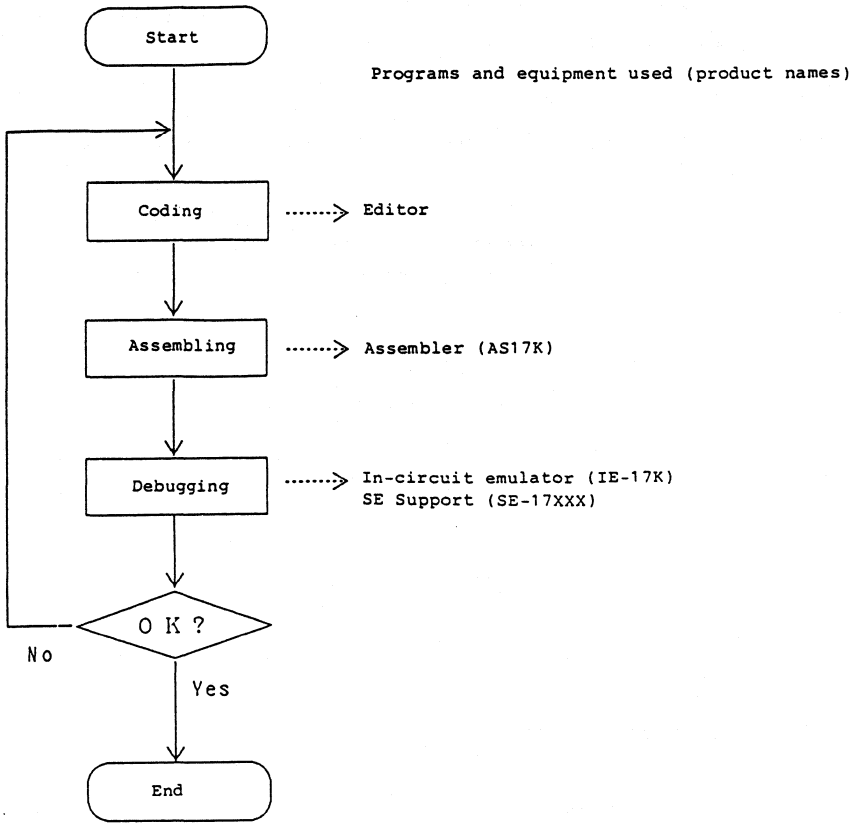


Figure 1.2 Software Development Sequence





### 1.1.5 Comparison of assemblers

Table 1.1 shows the features of both absolute and relocatable assemblers.

Table 1.1 Comparison Table of Assemblers

		Absolute assembler	Relocatable assembler
Assemble format		Batch assemble (however, the AS17K enables quasi-split assembling)	Split assemble ↓ link
Assemble list address display		Absolute address	Relative address
Variables	Operand section variable; operation limited	None	Limitations created by linker
	Local variables	Cannot be defined (however, definable with the AS17K)	Definable
Others		With batch assemblers, assemble time cannot be saved even though only one section of a source is amended. (However, since the AS17K enables quasi-split assembling, it may be possible to speed assemble times depending on the programming.)	Address calculation is necessary when debugging. Since split assembling is possible, the programming of a module can be carried out by a number of people.

## 1.2 An Overview of Assembler Functions of uPD17000

### 1.2.1 Generating sequence files

The AS17K assembler for the uPD17000 series is an absolute assembler. However, it is an absolute assembler which offers module programming, a feature of the relocatable assembler. There is no linker program such as a relocatable assembler package generally contains, so the AS17K is also provided with a link function.

When source modules are split and programmed, it is necessary to have a sequence file in which may be entered the sequence of the series of source module files and so forth. The sequence file also determines options at assemble time. If a source module consists of one file only, assembling will nevertheless take place even though there is no sequence file.

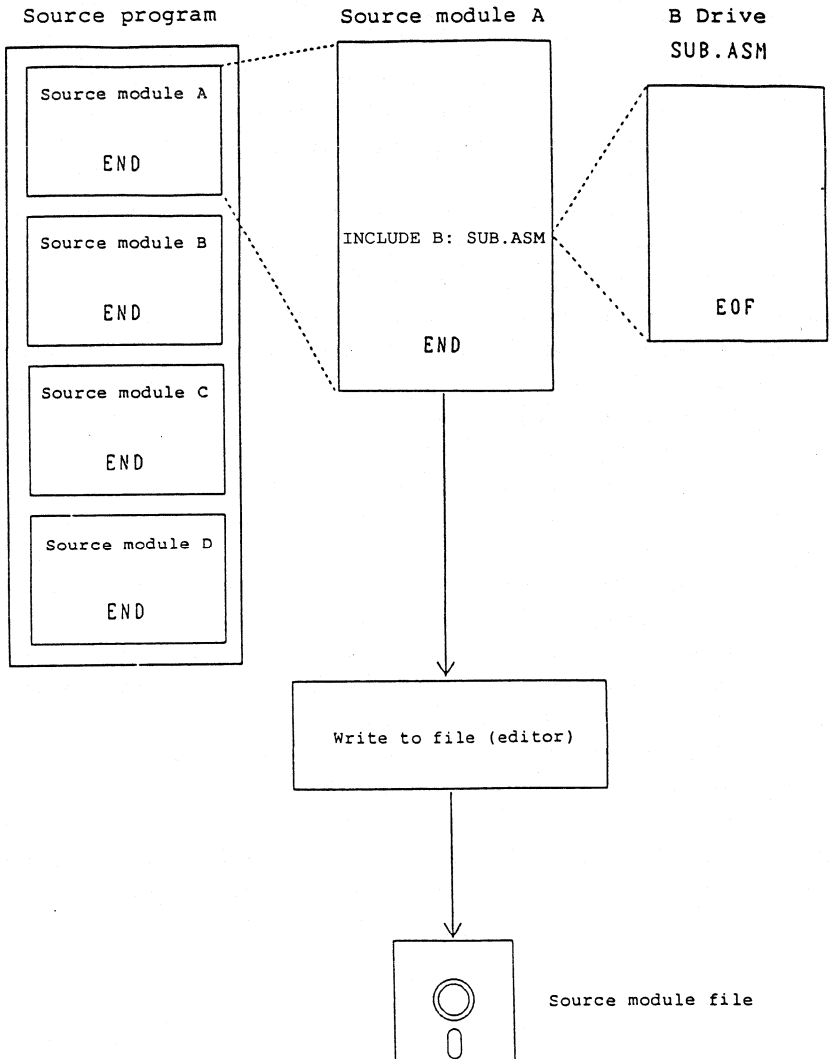
Also necessary when assembling is a device file which contains data specific to devices. This device file defines such data as instructions usable or the size of ROM and RAM for each device. The AS17K refers to this device file when assembling. A separate device file is prepared for each product in the series.

### 1.2.2 Generating source module files

Programs are generally designed so that they are split into sub-programs for each function. If sub-programs have a high degree of functional independence, debugging is easy; further by-products are greater efficiency in development and easier subsequent maintenance. A single sub-program is both a coding unit and an assemble input unit. Assemble input units are known as source modules. When source module coding is completed, the module is edited and so forth, and written to a file. The file so created is known as a source module file.

When source programs are split, it is necessary to have a sequence file in which the inter-relationships of the parts are entered. The sequence of a series of individual source modules, for example, may be entered in the sequence file. It is to be noted that the split source modules we are talking about here are different from split files created with the INCLUDE statement. A file designated by the INCLUDE directive can be regarded as part of the source module which contains that INCLUDE directive.

Figure 1.3 Generating a Source Module File



### 1.2.3 Assembling

It is necessary to have the following files in order to assemble a source module:

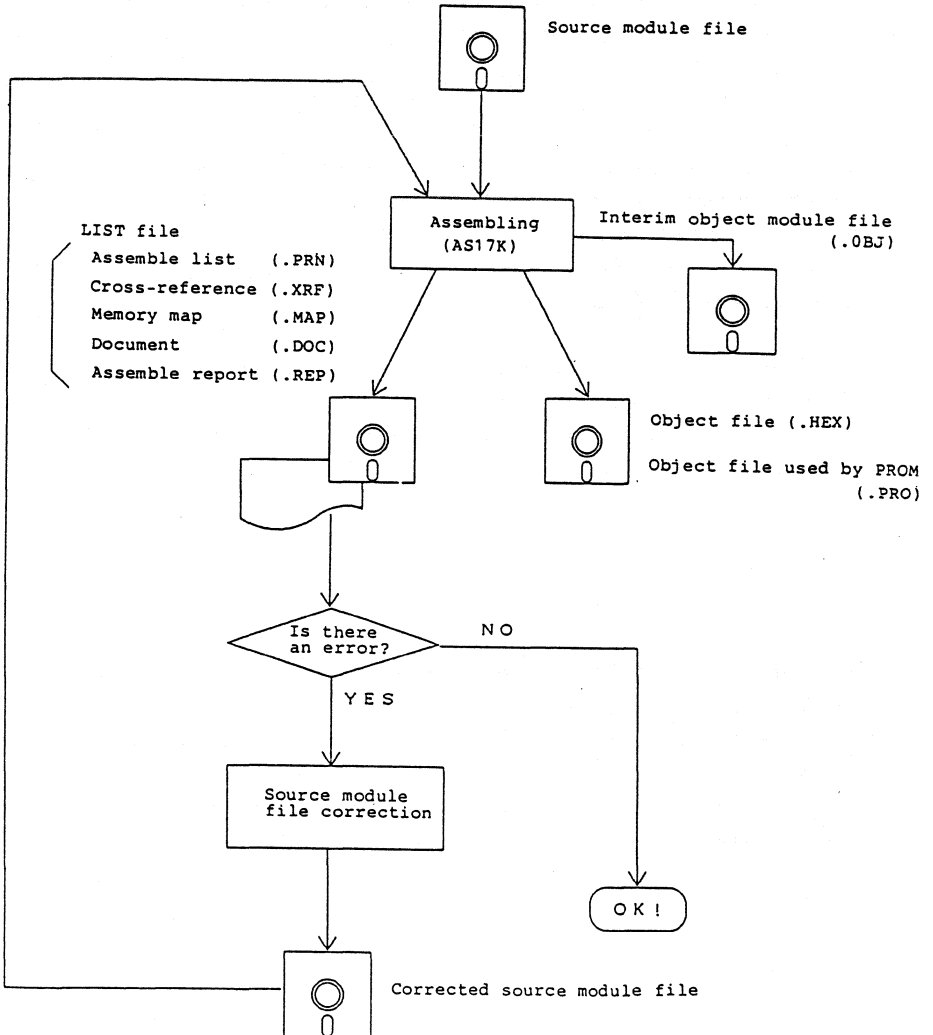
- an assembler file (AS17K. EXE)
- a device file (D17001. DEV, etc)
- a source module file (oooo. ASM, xxxx. ASM, etc)
- a sequence file (oooo.SEQ)

Output list control is carried out directly from the console when the AS17K is booted up, or by designating an assemble option in a sequence file. If an error is detected, for example in an assemble list, the source module should be corrected and assembling repeated until the error does not occur. If no errors occur, an object module file may be generated.

When a source program is split into modules, the AS17K generates an interim object module file when assembling (.OBJ). This interim object module file may be used when reassembling to carry out partial source program modification.

In order to reduce assemble time, the AS17K will only assemble a corrected source module; with uncorrected source modules, use the interim object module file which has already been generated. To tell whether a correction has been made or not, compare the times of generation of the source module file and the interim object module file of the same name; if the date of creation of the source module file is later, then it can be assumed that it has been corrected. Thus, in cases where there is no interim object module file, or where the source module file is earlier, the assembler automatically determines that this is so after generating the interim object module file, and assembles.

Figure 1.4 Generating an Object File



### 1.3 Before Beginning Program Development

This chapter sets out a few things which you need to know in advance in order to use the AS17K easily. Detailed explanations will be given in subsequent chapters.

#### 1.3.1 Restrictions on symbols

##### (1) Number restrictions

The symbol table region which may be used in one source module is 64K bytes.

With the AS17K, a maximum of 255 characters can be defined in one symbol. The number of symbols which can be used is as follows:

- where all symbols are 255 characters in length, 240 individual symbols may be set;
- where all symbols are eight characters in length, 3368 individual symbols may be set.

##### (2) Type limitations

A type must also be defined when defining a symbol. This type is defined by the symbol definition directive.

There are four varieties of type: data types (DAT type), data memory address types (MEM type), flag types (FLG type), and label types (LAB type). The relationship between type and symbol definition directives is given in Table 1.2.

Table 1.2 Correspondence between Type and Symbol Definition Directives

Type	Symbol definition directives
Data type (DAT type)	DAT
Data memory address type (MEM type)	MEM
Flag type (FLG type)	FLG
Label type (LAB type)	LAB

When carrying out an arithmetic operation with symbols of differing types, the operation should be executed after carrying out a type conversion. Further, since types which can be processed by mnemonics are limited, there will also be occasions on which it is necessary to carry out a type conversion while programming.

Defining types will allow the incidence of bugs when programming to be reduced, and further permits the documentation generation function which the AS17K possesses to be used more efficiently.

### (3) Symbols in macros

Symbols which cannot be globally declared are handled as local symbols.

#### 1.3.2 Restrictions on directives

Forty nesting levels are possible for the statements MACRO, REPT, IRP, IF and CASE. It is necessary that it be noted that this level will fall if developing a separate directive within a directive definition. Built in macros are also counted within the above-mentioned 40 levels. For nesting levels with built-in macros, please refer to Section 3.2.9 on built-in macro directives. Note that it is possible to refer to, but not define, a macro within a macro.

## UMAS17K ASSEMBLER

---

Directives which may be nested.

```
REPT  ~ ENDR
IRP   ~ ENDR
IF    ~ ELSE~ ENDF
CASE ~ EXIT~ OTHER~ ENDCASE
INCLUDE*
```

\* Nesting with the INCLUDE statement has eight possible levels, and is independent of the directives mentioned above.

### 1.3.3 Similar reserved words

Below are listed reserved words which have similar names; care should be exercised not to confuse them when designing programs.

#### (1) SETn and SET

SETn is a built-in macro instruction, while SET is a symbol definition directive. A completely different definition is achieved by incrementing n ( $1 \leq n \leq 4$ ).

#### Example 1

```
Flag A FLG 0.10H. 1 ] ①
Flag B FLG 0.10H. 2 ]
                               SET2 Flag_A, Flag_B ; ②
```

#### Description

- ① The addresses and bit positions of Flag\_A and Flag\_B are defined by the FLG directive.
- ② Flag\_A and Flag\_B are set by the SET2 built-in macro instruction.



### Example 2

Memory_1	MEM	0.40H	}	①
Flag_1	FLG	0.10H.2		
Label_1	LAB	2FFH		
	:			
Memory_1	SET	1.20H	}	②
Flag_1	SET	1.0FH.1		
Label_1	SET	7FH		

### Description

- ① The memory\_1, flag\_1 and label\_1 names are defined by the MEM, FLG and LAB directives.
  - ② The values assigned to the names may be altered by the SET symbol definition directive.
- (2) SKTn, SKFn and SKE, SKNE, SKGE, SKLT.
- SKTn and SKFn are built-in macro instructions, while SKE, SKNE, SKGE, and SKLT are instructions (mnemonics) for the UPD17000 series devices themselves.

```
Flag_1  FLG  0.10H.1  
Flag_2  FLG  0.10H.2  
Memory_1 MEM  0.20H  
.  
.  
SKT2 Flag_1, Flag_2 ; ①  
.  
.  
SKE Memory_1, #01H ; ③  
.  
.
```

## (Explanation)

- ① The names of Flag\_1, Flag\_2 and Memory\_1 are defined by the FLG and MEM symbol definition directives.
- ② When the statuses of Flag\_1 and Flag\_2 are tested and both are set with the SKT2 built-in macro instruction, the following instruction is skipped.
- ③ The SKE instruction is used to determine whether the contents of memory\_1 is 1, and if so, the following instruction is skipped.

### 1.3.5 Setting the time and date of the host

The current time and date should always be checked when booting up PC-DOS on the host IBM PC-AT.

When assembling the AS17K carries out a comparison of the dates of generation of interim object module files which have the same names as source module files. If, as the result of this comparison, the time of generation of the object module file is found to be later than that of the source module file, that source module will not be assembled.

If the time on the host clock is later than the time of generation of the source module file, notwithstanding the fact that the source module file may have been amended, the results of assembling shall always reflect the status of that file before it was amended.

### 1.3.6 Restrictions on the number of source modules

With the AS17K, it is possible to split the module into a maximum of 99 source programs and program it.

The split source module can be handled as a single source program by entering assemble processing order sequences to a sequence file (.SEQ).

#### 1.4 Features of the Assembler

This section introduces the features of the AS17K.

##### 1.4.1 pc-DOS assemblers

The AS17K operates in the IBM PC/AT with PC-DOS (Version 3.1).

##### 1.4.2 Capacities for program modularization

A relocatable assembler requires a program known as a linker in order to combine modularized programs. That is to say, a relocatable assembler carries out assembling with two programs, the assembler and the linker.

The AS17K is an absolute assembler, but it is provided with a program modularization function which is characteristic of relocatable assemblers. In order to assemble multiple source modules, the AS17K must have a file in which is entered the names of the modules and the order in which they are to be assembled. This is called the sequence file (.SEQ). A sequence file may contain various types of conditions pertinent to assembling apart from the names of the source modules.

The AS17K is provided with an assemble time reduction function in order to assemble modularized source programs more efficiently. When the AS17K assembles, a comparison is made of the source module and the interim object module file with the same name. If, as a result of this comparison, the time of generation of the source module is found to be later, that source module is assembled a second time. If the time of generation of the source module is earlier, it is judged to be a module which has not been changed, and assembling will not take place. Accordingly assembly time can be greatly reduced depending on how the debugging goes. Please refer to Part 2 Section 6.4 on methods of reducing time spent debugging for more on this.

##### 1.4.3 Convenient built-in macro instructions

The AS17K has a built-in macro instruction, the purpose of which is to increase the efficiency of programming and make programs easier to read. The use of the built-in macro instruction is recommended for setting, re-setting, inverting or initializing

flags, or skipping and switching memory banks in accordance with a flag. It is an extremely effective instruction for creating program libraries. The built-in macro instruction can also be used for quick development of user-defined macro instructions. The amount of time required for assembling when developing a built-in macro instruction is about the same as the time required for converting to a mnemonic machine language.

### 1.4.5 The documentation generation function

It is possible, with the AS17K, to enter documentation into source programs by using the documentation generation instructions SUMMARY and TAG. This instruction can be used at the beginning of a program module or routine to insert a description of how it operates or the design procedure. If, during assembling, documentation generation is specified in a sequence file, it is possible to generate and extract documentation separately from assembler lists. Documentation may be generated for symbol lists and summaries which are used in program modules and routines. When generating source programs, the time taken for the task of producing documentation after program design is completed can be greatly reduced by entering program explanations in Japanese using the documentation generation control instruction.

It is also possible, with the AS17K, to generate automatically memory maps and flag maps effectively by using the symbol definition directive. For more details, please refer to part 2 section 4.5.11 on the map file output control option.

### 1.4.6 Two types of cross-reference functions

The AS17K has cross-reference functions with the following two types of formats:

**(1) Table format**

At the end of an assemble list, cross-references may be output in alphabetic symbol order.

**(2) Buried formats**

A cross-reference list may be generated in the label definition section of an assemble list. De-bugging can be carried out most efficiently as statement addresses which are referenced in the assemble list can be displayed.

**1.4.7 The assemble report function**

The AS17K assemble report function will generate an assemble report file (.REP) which will output:

- (1) The time required for assembling
- (2) The memory/file volume used
- (3) The number of macros used
- (4) The number of public and local symbols
- (5) The list and transaction drive created when modules are assembled
- (6) The file generated when linking
- (7) The number of errors and warnings generated.

It is possible to carry out tasks more efficiently by checking the assemble report file.

**1.4.8 The automatic object load function**

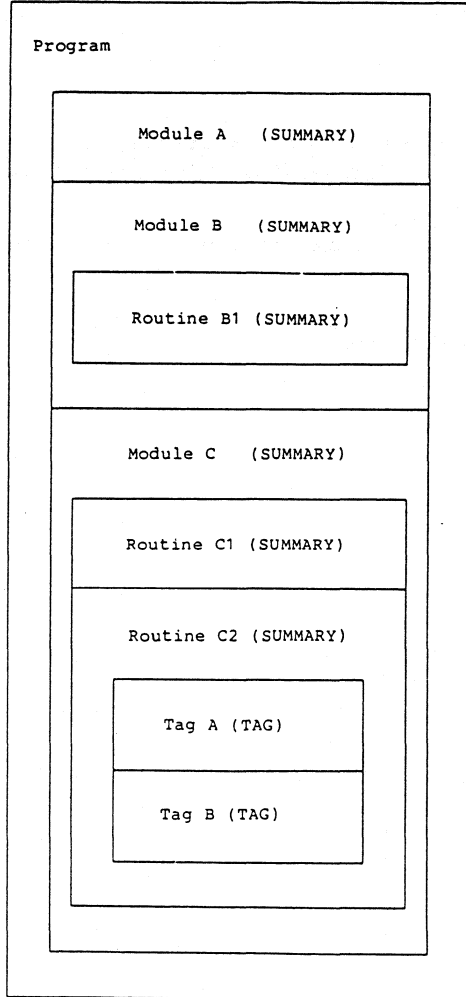
When the host IBM PC-AT is connected to the IE-17K in-circuit emulator, it is possible, while assembling, to download the object codes determined automatically to the IE-17K.

**1.4.9 Source program hierarchy creation**

The AS17K will permit hierarchial programming through the effective use of the source program module splitting function, and the documentation generation control instructions SUMMARY and TAG.

Figure 1.5 Hierarchical Programming

Source program



Hierarchical levels in programs are, from the top:

1. Program
2. Module
3. Routine
4. Tag

This method of hierarchical programming is very effective when debugging using the simple host, which is one of the uPD17000 series development tools.

For details please refer to the PD17000 series simple host manual.\*

\*: To be published



### CHAPTER 2 METHODS OF ENTERING SOURCE PROGRAMS

#### 2.1 The Basic Operations of Source Programs

Source programs are, as shown in figure 1.3, composed of source modules. Source modules are composed of statements. For some notes on the structure of statements, please refer to section 2.2 on the structure of statements.

There are no limitations on the size of a source module. Thus, there are no limitations on the number of statements which may be entered. However, the maximum number of modules possible in a split source module is 99.

Instructions, directives and control instructions may be entered into a source program in the position desired; however, it is necessary to enter an END directive and only that instruction at the end of each source module.

## UMAS17K ASSEMBLER

### 2.2 The Structure of a Statement

An assembler language source program is composed of statements. A statement is entered using the characters specified in section 2.4 on the character set.

When source programs are generated using a text editor, each statement is terminated by a carriage return or a line feed, but the assembler will only interpret a line feed as the end of a statement and will ignore a carriage return.

As explained below, a statement is composed of four fields: Symbol field, Mnemonic field, Operand field and Comment field.

Each field is delimited by a space 8-bit JIS code 20H, a TAB (09H), a colon (:), (3AH), or a semicolon (;) (3BH). The number of characters in a line is unlimited, the end of the statement being determined by a line feed.

With free entry formats, a statement may be written from any field, so long as the order is Symbol field, Mnemonic field, Operand field, Comment field.



- ① Symbols entered in the Symbol field are delimited by a colon or a space created by a Blank or TAB character. Colons and blank spaces are different when using instructions for entering to the Mnemonic field.
- ② If an Operand field is required, it is delimited by spaces.
- ③ Comments entered in the Comment field are delimited by semicolons.
- ④ As many spaces as are desired may be entered before and after colons and semicolons. Example 1 below illustrates the situation in which a colon is entered between a Symbol field and a Mnemonic field, while example 2 indicates what occurs when a blank space is used.

(Example 1)

```
AAA : LD  REG, MEMORY    ; Load memory to register
BBB : ST  MEMORY, REG    ; Store register to memory
```

(Example 2)

```
AAA  SET  3
BBB  DAT  5
```

2.3 The Tabulation Function

The AS17K is provided with a tabulation function in order to put assemble lists into formats which are easy to read. The tabulation function organizes variously Symbol fields, Mnemonic fields, Operand fields and Comment fields in source programs from each eighth field.

(Example)

Addition:

```

ADD      REG1, MEM1
ADD      REG2, MEM2      ; 8-bit addition
    
```

Column divisions in multiples of eight integers (number of tabulations)

To operate the tabulation function, a TAB (horizontal TAB, 09H) is inserted before the semicolon which indicates the start of a Mnemonic field, Operand field, or Comment field in a source program.



The AS17K has the capability for deciding, depending on the printer being used, whether to send a TAB code (09H) or a blank code to fill in. This is provided in order to deal with printers which cannot recognize TAB codes. In this situation, the AS17K can designate a blank code to be sent to the printer in place of the TAB code. It is recommended, when using the simple host, that TAB codes be used in order to use disk space efficiently. For details, please refer to the uPD17000 series simple host manual.

### 2.4 The Character Set

Eight-bit JIS code characters and shift JIS code characters (refer to Appendix 8) should be used to enter statements. There are restrictions on the use of characters as symbols. For details, please refer to the rules given in section 2.5.2 on entering symbols. Note that with reserved words, there is no distinction between lowercase characters and uppercase characters.

(Example 1)

AAA	DAT	3
AAa	DAT	5

AAA and AAa are interpreted as different symbols.

(Example 2)

MOV	MEM1,#1
Mov	mem1,#3

MEM1 and mem1 are different symbols. MEM1 is set as one, while mem1 is set as three. However, the reserved word MOV is interpreted as the same as Mov.

#### 2.4.1 Alphanumeric characters

The alphanumeric characters consist of alphabetic characters and numeric characters.

#### 2.4.2 Numeric characters

The binary numerals are the numerals 0 and 1.

The octal numerals are the numerals 0, 1, 2, 3, 4, 5, 6 and 7.

The decimal numerals are the numerals 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

The hexadecimal numerals are the numerals 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

### 2.4.3 The use of special characters

It will be more efficient if en characters are used in the situations given below. If em characters are used in these situations, the applications cannot be used, and the characters will be interpreted simply as characters. In character strings (characters and constants) or Comment fields, these special characters may be used to signify themselves, line feeds excepted.

Symbol	Name	Main use
	Space	Spaces delimit fields
?	Question mark	Character equivalent to alphabetic character
@	Unit price symbol	A symbol used for bit addressing
_	Underscore	Character equivalent to alphabetic character
,	Comma	Delimiter between operands
.	Period	Decimal point or a symbol used to indicate the value of the location counter
+	Plus sign	Positive sign or ADD operator
-	Minus sign	Negative sign or SUBTRACT operator
*	Asterisk	MULTIPLY operator
/	Slash	DIVIDE operator
(	Left parenthesis	A symbol used in pair with a right parenthesis to change the order of precedence in operations or for addressing
)	Right parenthesis	Ditto
\$	Dollar mark	Location counter value
=	Equal sign	Comparative operator
;	Semicolon	Indicates the beginning of a comment
:	Colon	Delimits labels
'	Single quotation mark	A symbol used to indicate the beginning or end of a character constant
<		Comparative operator
>		Ditto
#	Sharp sign	A symbol used to indicate the value of an immediate data

&	Ampersand	Indicates a character string linkage within a macro
TAB code		A character corresponding to eight blank spaces
LF code		Indicates the end of a statement
CR code		Not recognized by the assembler
NULL code		Ditto
FF cod		Ditto

## UMAS17K ASSEMBLER

---

### 2.5 The Symbol Field

Symbols are entered in the Symbol field. When a symbol is entered in the Symbol field, that symbol is said to be defined. Symbols may be classified as labels or names, depending on their purpose and how they are defined.

#### (1) Names

A symbol which is defined by the directives DAT, SET, MEM, FLG, or LAB is known as a name. Names are allocated to numeric data or addresses. They substitute for these numeric data or addresses so that the name defined can be used in a program. That is to say, numeric data cannot as it is be handled; it has to be given a name when it is desired to use them. For example, if the data memory (RAM) 00H address is named "REGO"; when that address needs to be used the name "REGO" is used. In this situation, "REGO" is referred to as a name.

(Example)

REGO	MEM	0.00H
OCTOBER	DAT	10H

The 00 (BANK0) address in data memory is defined as the name REGO.

The numeric data 10H is defined as the name OCTOBER.

#### (2) Labels

Labels are symbols which are allocated to the ORG, DW or DB directives, or instruction (mnemonic) addresses. They are used to refer to the program memory addresses (location counter values) allocated to the instruction or directive to which they are attached.

That is to say, a label is attached to the first address in a routine as a name giving some sort of indication of what that routine does, and is used when referring to or branching to that routine from another routine.



(Example)

Address		
0010	SUBROUTINE: ADD	MEMORY1, #3
0011	ST	MEMORY2, REGISTER
:		
0030	BR	SUBROUTINE
:		

In this example "Subroutine" is referred to as a label.

### 2.5.1 Symbol types

All symbols may be classified into one of the following for types:

1. Data type (DAT type) — a symbol which defines a constant
2. Data memory address type (MEM type) — a symbol which defines a data memory (RAM) address;
3. Flag type (FLG type) — a symbol which defines a flag (one bit of a address in RAM);
4. Label type (LAB type) — a symbol which defines a program memory address (location counter value).

### 2.5.2 Rules for entering symbols

The rules for entering symbols are as given below.

- (1) The characters used in symbols are 8-bit code characters and shift JIS code characters other than the special characters. (except for "\_", "?") The special underscore and question mark characters are used. A en numeric character may not be used as the first character. However, numeric labels entered into CASE END CASE directives may use en numeric characters.
- (2) The length of a symbol is between one and 255 characters, in the case of en characters. More than 256 characters may be entered in a Symbol field, but only the first 255 from the beginning will be effective.
- (3) Labels are terminated with a colon (3AH). A space or a tabulation code (TAB) may be inserted between the label and the colon.

- (4) A name must always be entered in the Symbol field when using the DAT, SET, MEM, FLG, LAB and MACRO directives. Names are terminated by a space or a tabulation code (TAB).
- (5) It is not possible to define the same symbol more than once. If this is done, an S error (symbol multi-defined) will be generated. However, symbols defined by a SET directive, or symbols which have not been globally declared and defined in a macro, are exceptions to this. If the symbol is not declared publicly, it is possible to use the same symbol in another module. On these occasions, these symbols will be treated as different symbols.
- (6) A reserved word may not be defined as a symbol.

(Example 1)

Correct example

F1F4:  
 LABEL:  
 HERE:  
  
 ANH:  
  
 ENDX:

Wrong example

1F4F: ... Started with a numeral  
 LABEL: ... No colon attached  
 HE RE: ... Blank space occurs within  
           a symbol  
 AND: ... Cannot be used by the  
           instruction  
 END: ... Cannot be used by the  
           directive

(Example 2)

In ABC .... XYZ: where X is the 255th character the symbol will be interpreted as ABC ..... X:.


(Example 3)

ABC DAT 3  
  
 XYZ DAT ABC

Three is assigned similarly to ABC and XYZ.

(Example 4)

```
LOOP: MOV  RO,#20H  
      :  
LOOP: ADD  RO,#20H  
      :  
      BR   LOOP
```

A diagram consisting of a vertical line on the right side of the code block. At the top of this line is a horizontal line that points to the right, ending at the first 'LOOP:' label. At the bottom of the vertical line is a horizontal line that points to the left, ending at the 'BR LOOP' instruction. This diagram illustrates that the branch instruction jumps back to the first definition of the symbol 'LOOP'.

The symbol LOOP is defined twice.

On the second occurrence of the definition, an S error (symbol multi-defined) is generated.

If a symbol is defined twice, the effective definition will be the first one.

## UMAS17K ASSEMBLER

---

### 2.6 The Mnemonic Field

Instructions, directives and console instructions are entered in the Mnemonic field. With instructions which require operands, one or more space codes or tabulation codes (TAB) are required in order to distinguish between the Mnemonic field and the Operand field.

(Example)

Correct example

BR LOOP

RET

ADD M,#1

Wrong example

BRLOOP ... There is no space between the Mnemonic and Operand fields

RE T ... A space has been inserted within a mnemonic

AD M,#1 .. AD does not occur in uPD17000 series instructions.

### 2.7 The Operand Field

Data (operands) which are necessary for the execution of instructions are written in the Operand field. There are instructions which do not need operands, as well as instructions which require one, two or more operands, such as macro directives. Where two or more operands are required, each operand is discriminated by a comma. One or more spaces or tabulation codes (TAB) are required to be inserted between a Mnemonic field and an Operand field.

#### 2.7.1 Entry format for the Operand field

There are five formats, as given below, for entry to Operand fields.

##### (1) Constants

A constant may be a numeric constant composed entirely of numerals, or a character constant composed of characters. Numeric constants may be binary, octal, decimal or hexadecimal; they are entered in en characters.

##### (a) Binary constants

Binary constants are indicated by binary strings with the en character B added at the end.

Example 1011B

##### (b) Octal constants

An octal constant is indicated by an octal character string with en character O or Q attached at the end.

Example 730

73Q

##### (c) Decimal constants

A decimal constant is indicated by a decimal character string with en character D or nothing attached to the end.

Example 927

927D

## (d) Hexadecimal constants

A hexadecimal constant is indicated by a hexadecimal string with an H character attached at the end. Where the first character is an H character other than 0 to 9, 0 is attached at the beginning.

Example 9CH

0ABH (If ABH is entered, this is interpreted as a symbol).

## (e) Character constant

Character constants are composed of 8-bit JIS code characters (line feed excepted) or shift JIS code characters enclosed in commas.

Characters enclosed in commas will, as a result of assembling, be converted to 8-bit JIS codes or shift JIS codes.

If using commas as character constants, two commas should be used one after the other. Character constants cannot be operated on.

(Examples)

'A' ...41H

(en)

' A' ...8360H

(em)

'''' ...27H

(en) A single comma is reserved as a constant.

' A ''' ...4127H

(en )

' ' 20H

(en space)

' <' ...203CH

## (2) \$ (location counter)

The \$ sign indicates a location counter value. Thus, where it is used it gives the program memory address of that instruction.

(Example) Address

```
100          MOV    R0,#20H
101 LOOP:    ADD    R2,#30H
102          BR     $-1
103          BR     $+20H
```

The \$ in "BR \$-1" indicates the address 102H. Thus, \$-1 indicates the address 101H. The \$ in the example "BR \$+20H" indicates the address 103H. "BR \$-1" uses a label and operates similarly to "BR LOOP".

### (3) Symbols

Where a symbol is entered in an Operand field, the value assigned to that symbol (label or name) is regarded as the operand value.

(Example 1)

Here:	BR	There
	:	
There:	RET	

(Example 2)

VALUE DAT 1H
ADD RO,#VALUE

"ADD RO, #VALUE" has the same meaning as "ADD RO, #1H".

### (4) Expressions

When the constants, dollar signs or symbols mentioned above are linked by operators, they are known as expressions. There are 17 types of operators (+, -, \*, /, MOD, NOT, AND, OR, XOR, SHR, SHL, EQ or =, NE or <>, CT or >, GE or >= LT or <, and LE or <=). The order of priority of execution of operations is fixed.

The bit position segment symbol required for writing memory or flag addresses may also be entered as part of an expression. For more details, please refer to section 2.9 on expressions and operators.



### 2.8 The Comment Field

The Comment field begins with a semicolon, and the comment follows after. Comments are notes written in to help in the understanding of the contents of programs when referring to assemble lists; they are output with the assemble list, but are ignored by the assembler.

If two semicolons are used together in a macro definition, the assembler will treat them as a comment within the macro definition, and they will not be printed out when the macro is developed.

(Example)

```
HERE:MOV OOH,1 ; THIS IS A COMMENT  
;  
; BEGIN LOOP HERE  
;
```

If the character following the semicolon(;) is a period(.), the character string following the period is registered as a TAG. (Please refer to section 3.3.4 on the Documentation Generation Control Instruction).

## 2.9 Expressions and Operators

## 2.9.1 Expressions

Character or numeric expressions displayed in the Operand field using symbols, constants or operators are known as expressions. Expressions are of four types: data type (DAT type) expressions, data memory address type (MEM type) expressions, flag type (FLG type) expressions and label type (LAB type) expressions.

An explanation of the method of generating these different types of expressions is given below.

For information on the symbol types used in these expressions, please refer to section 3.2.2 on Symbol Definition Directives.

## (1) Data type (DAT type) expressions

Data type expressions are used when representing 16-bit data. When the result of operating on an expression is 17 bits or more, a V error (illegal operand value) is generated.

However, with data type expressions which are entered as the operands of instructions, immediate data which has a # attached immediately in front of the expression indicates 4-bit data. In this situation, if the result of the expression is five bits or more, a V error will also be generated.

It is possible to use constants or symbols which have already been defined as data types. In a data type expression, symbols other than data types may be used in data type expressions, but type conversion must be effected.

(Example)

```

count_number DAT 0256H ; ①
MEM1 MEM 0.00H ; ②
:
MOV MEM1,#count_number/82H ; ③
:
ADD MEM1,#count_number*4H ; ④
      ↑
      ; V error generated

```

(Comments)

- ① The value 0256H is allotted to the name count number.
- ② The bank 0 data memory address 00H is allotted to the name MEM1.
- ③ The count number/82H ( $256H/82H = 4H$ ) is stored in MEM1. Count number/82H is a data type expression.
- ④ In this example, count number\*4H executes  $256H \times 4H$ ; because the result of the operation is in five bits or more, a V error is generated.

(2) Data memory address type (MEM type) expressions

An MEM type expression is used to represent a data memory address. MEM type expressions are able to use the position segment symbol ".". After execution of the operation, only the lower 12 bits of data are significant.

Symbol types which are able to make use of MEM type expressions are MEM types and DAT types.

(Example)

MEM4	MEM	0.10H	
MEM5	MEM	0.20H	
CONST1	DAT	2H	
CONST2	DAT	4H	
	:		
	MOV	MEM4+4H, #CONST1	; ②
MEMA	MEM	CONST1+3H .CONST2+2H	; ③
	:		

## UMAS17K ASSEMBLER

(Comments)

- ① MEM4, MEM5, CONST1, and CONST2 are defined by a symbol definition directive.
- ② The expression MEM4+4H means the bank 0, data memory address 14H. MEM4 is an MEM type symbol.
- ③ The expression CONST1+3H. CONST2+2H refers to the bank 5 data memory address 06H. Thus, MEMA defines the bank 5 data memory address 06H.

CONST1 and CONST2 are DAT type 5 symbols.

### (3) Flag type (FLG type) expressions

A flag type expression is used to represent a flag. In a flag expression, a flag type parallel operation cannot be executed. Only operations within the range delimited by position segments indicators (.) are effective. The symbols which may be used are DAT type and MEM type.

Example:

MEM6	MEM	0.13H	}	①
CONST3	DAT	0H		
CONST4	DAT	14H		
CONST5	DAT	3H		
:				
FLAG1	FLG	MEM6.0H		②
FLAG2	FLG	CONST3+2H.CONST4+6H.CONST5		③

(Explanation)

- ① MEM6, CONST3, CONST4 and CONST5 are defined by the symbol definition directive.
- ② Bank 0 data memory address 13H(MEM6) and bit position zero (LSB) are assigned to the name FLAG1. In this situation, MEM6 is an MEM type symbol.
- ③ The bank address 2 data memory address 1AH and bit position 3 (MSB) is assigned to the name FLAG2. In this situation, CONST3, CONST4 and CONST5 are DAT type symbols.

### (4) Label type (LAB type) expressions

An LAB type expression is used to represent the value of a program memory address (location counter).

Program memory addresses (location counters) may be represented by up to 16 bits.

An LAB type expression is capable of being used as a symbol defined by an LAB expression and a constant.

If a symbol other than an LAB type symbol is used in an LAB type expression, it is necessary to carry out a type conversion.

Example:

```
Data_table_1    LAB    0300H                ; ①
                :
                ORG    Data_table_1        ; ②
Table_area_1:   DB     00H,48H
                :
                ORG    Data_table_1+20H   ; ③
Table_area_2:   DB     10H,52H
                :
                ORG    Data_table_1+40H   ; ④
Table_area_3:   DB     50H,60H
```

(Explanation)

- ① The value 0300H is allocated to the name data\_table\_1.
- ②③④ The first address in the table region is defined by a label type expression.

2.9.2 An outline of operators

(1) Outline

There are five types of operator in the AS17K assembly language; the order of priority of operations is fixed.

- ① Arithmetical operators  
+, -, \*, /, MOD
- ② Logical operators  
OR, AND, XOR, NOT
- ③ Comparative operators  
EQ, NE, LT, LE, GT, GE  
=, < >, <, <=, >, >=
- ④ Shift operators  
SHR, SHL
- ⑤ Others  
( ) (symbol designating the order of operations).

(2) The order of priority of operators

The order of priority of operators is fixed as in the following table, but it is possible to alter this order by using parentheses. If operators of the same order of priority occur in an expression, the operation takes place from the left. In the following table, the highest order of priority is given as 1.

Table 2.1 The Order of Priority of Operators

Order of priority	Operator
1	( ) (Symbol indicating order of operation)
2	*, /, MOD, SHL, SHR
3	+, -
4	EQ, NE, LT, LE, GT, GE =, < >, <, <=, >, >=
5	NOT
6	AND
7	OR, XOR

### 2.9.3 Arithmetical operators

#### (1) Addition operators

##### [Format]

<expression 1>+<expression 2>, +<expression 3>

##### [Function]

Adds the value (evaluation) of the expressions entered on either side of the operator.

##### [Explanation]

If the result of the operation, including the symbol bits, exceeds the range of 16 bits ( $-2^{15}$  to  $+2^{15}$ ), a V error (initial valid value) will be generated.

##### [Example]

```
START DAT 4H
OFFSET DAT 3H
STEP DAT 2H
R1 MEM 0.01H
:
MOV R1,#START+OFFSET ; ②
LOOP1:
ADD R1,#STEP ; ③
SKF1 CY ; ④
BR LOOP1END
:
BR LOOP1
LOOP1END:
```

##### (Comments)

- ① Defines symbol.
- ② Stores START+OFFSET(07H) to R1 as the initial value.
- ③ Adds STEP to R1.
- ④ If there is a carry, jumps to LOOP1END.

(2) Subtraction operators

[Format]

<expression 1>-<expression 2>, -<expression 3>

[Function]

Subtracts the value (evaluation) of the expression on the right from the value of the expression on the left of the operator.

[Explanation]

If the result of the operation, plus the symbol bits, exceeds the range of 16 bits ( $-2^{15}$  to  $+2^{15}$ ), a V error (invalid value) will be generated.

[Example]:

```

TABLE_end LAB 100H ] ①
TABLE_area LAB 40H ]
:
ORG TABLE_end - TABLE_area ; ②

TABLE_start:

DW 0445H ] ③
DW 5637H ]
:
ORG TABLE_end
:
    
```

(Comments)

- ① Defines a symbol.
- ② The table start address is taken as "TABLE\_end - TABLE\_area" (0C0H).
- ③ Effects a data definition.



### (3) Multiplication operators

#### [Format]

<expression 1>\*<expression 2>

#### [Function]

Multiplies the values of the expressions (evaluations) entered on either side of the operator.

#### [Explanation]

If the result of the operation, plus the symbol bits, exceeds 16 bits ( $-2^{15}$  to  $+2^{15}$ ), a V error (invalid value) will be generated.

#### [Example]

Table	LAB	100H	} ①
Block	LAB	10H	
	:		
	ORG	Table	} ②
	:		
	ORG	Table + block	
	:		
	ORG	Table + (block*2H)	
	:		
	ORG	Table + (block*3H)	

#### (Comments)

- ① Defines a symbol.
- ② Defines the first addresss of the table region in program memory.

(4) Division operators

[Format]

<expression 1>/<expression 2>

[Function]

Divides the value (evaluation) of the expression on the left of the operator by the value (evaluation) of the expression on the right of the operator.

[Explanation]

- o If the result of the operation, plus the symbol bits, exceeds the range of 16 bits ( $-2^{15}$  to  $+2^{15}$ ), a V error (invalid value) will be generated.
- o If the value of expression 2 is 0, the result of the operation will be 0.
- o If the expression is indivisible, the part below the decimal point will be discarded.

[Example]

```

Table area   LAB   40H
Table start  LAB   200H
:
              ORG  Table_start + (Table_area/4H)
:
              ORG  Table_start + (2*(table area/4H))

```

} ①

} ②

(Comments)

- ① Defines a symbol.
- ② Defines the first address in the table region.

### (5) The MOD operator

#### [Format]

<expression 1>ΔMODΔ<expression 2>

#### [Function]

Obtains the remainder when the value (evaluation) of the expression on the left of the operator is divided by the value (evaluation) expression on the right of the operator.

#### [Explanation]

If the value of the right expression is 0, the result of the operation will be 0.

#### [Example]

```
Constant_1 DAT 552H
Constant_2 DAT 7H
R1 MEM 0.10H
:
ADD R1, #Constant_1 MOD Constant_2
```

#### (Comments)

In the example given above, the result of executing "constant\_1 MOD constant\_2" will be 4H.

## UMAS17K ASSEMBLER

---

### 2.9.4 Logical operators

#### (1) The OR operator

[Format]

<expression 1> Δ OR Δ <expression 2>

[Function]

Returns the logical sum of the values (evaluations) of the expressions designated before and after the operator.

[Explanation]

Negative numeric values is processed as twos complements, the symbol bits, similarly, are operated on as numeric values.

[Example]

R1	MEM	1.40H
Constant_1	DAT	4H
	:	
	SUB	R1, #Constant_1 OR 8H

(Comments)

In the above example, the result of executing "constant\_1 OR 8H" is "0CH".

### (2) The AND operator

#### [Format]

<expression 1> Δ AND Δ <expression 2>

#### [Function]

Returns the logical product of the values (evaluations) of the expressions designated on either side of the operator.

#### [Explanation]

A negative numeric value is processed as twos complement; the symbol bits, similarly, are operated on as numeric values.

#### [Example]

```
Constant_1  DAT  4567H
            R10  MEM  2.50H
            :
            MOV  R10,#(Constant_1/2H) AND 0FH
```

#### (Comments)

In the abovementioned example, the result of executing "(constant\_1/2H) AND 0FH" is "0CH". " AND 0FH" is used because in DAT type expressions, the lower four bits only are designated as significant.

If in the above example, " AND 0FH" is omitted, an error will be generated.

## (3) The XOR operator.

## [Format]

<expression 1> Δ OR Δ <expression 1>

## [Function]

Returns the exclusive logical product of the value of the expression (evaluation) designated on either side of the operator.

## [Explanation]

Negative numeric values are processed as twos complements; the symbol bits also are operated on as numeric values.

## Example:

```
Constant_A DAT 2345H
Constant_B DAT 42H
      R02 MEM 0.42H
      :
      ADD R02,#((Constant_A-Constant_B) XOR 0FH) AND 0FH
```

## (Comments)

In the example given above, the result of executing " $((\text{constant\_A}-\text{constant\_B}) \text{ XOR } 0\text{FH}) \text{ AND } 0\text{FH}$ " is "0BH". " $\text{AND } 0\text{FH}$ " is "0CH". " $\text{AND } 0\text{FH}$ " is used because in DAT type expressions, the lower four bits only are designated as significant.

If, in this example, " $\text{AND } 0\text{FH}$ " is omitted, an error will be generated.

### (4) The NOT operator

#### [Format]

NOT  $\Delta$  <expression>

#### [Function]

Returns 1 complement of the evaluation of the expression designated.

#### [Explanation]

Negative numeric values are processed as twos complements; the symbol bits, also, are processed in the same way as bits defined as numeric values.

#### [Example]

```
Constant DAT 4567H
          R9 MEM 0.12H
          :
          MOV R9, #(NOT constant) AND 0FH
```

#### (Comments)

The result of executing "(NOT constant) AND 0FH" in the above example is "8H".

AND 0FH" is "0CH". " AND 0FH" is used because in DAT type expressions, the lower four bits only are designated as significant.

If, in the example given above, " AND 0FH" is omitted, an error will be generated.

## UMAS17K ASSEMBLER

### 2.9.5 Comparative operators

These operators carry out a comparison of the left and the right, and if the result is true, return -1; if false, they return 0 as the result of the operation.

#### (1) The EQ (Equal) operator

##### [Format]

```
<expression 1> ΔEQ Δ<expression 2>  
<expression 1>=<expression 2>
```

##### [Function]

If the values (evaluations) of the left and right expressions are the same, -1(true), is returned; if they are not the same, 0 (false) is returned as the result of the operation.

##### [Explanations]

- o It is possible to enter EQ as an = sign.
- o As -1 is processed as twos complements, in hexadecimal it is the value "OFFFH".

##### [Example]

Condition	DAT	OAH	] ①
R1	MEM	0.43H	
	:		
Macro	MACRO	P1,P2,P3	] ②
	IF	P1 EQ Condition	
	MOV	R1,#P2	
	:		
	ELSE		
	MOV	R1,#P3	
	:		
	ENDIF		
	ENDM		

##### (Comments)

- ① Defines a symbol
- ② Carries out a macro definition. P1, P2 and P3 are temporary parameters.  
If P1 = condition, IF~ELSE is developed; if P1 not = condition, ELSE~ENDIF is developed.



(2) The NE (Not Equal) operator

[Format]

```
<expression 1> Δ NE Δ <expression 2>  
<expression 1> < > <expression 2>
```

[Function]

If the left and right expressions are not equivalent, returns -1 (true); if they are the same, returns 0 (false) as the result of the operation.

[Explanation]

- o "NE" may be entered as "< >".
- o -1 is processed as twos complements; in hexadecimal, it is expressed as "OFFFh".

[Example]

Condition	DAT	OBH	} ①
R3	MEM	1.34H	
	:		} ②
Macro	MACRO	P1, P2, P3	
	IF	P1 NE Condition	
	MOV	R3, #P2	
	:		
	ELSE		
	MOV	R3, #P3	
	:		
	ENDIF		
	ENDM		

(Comments)

- ① Defines a symbol
- ② Carries out a macro definition. P1, P2 and P3 are temporary parameters. If P1 not = condition, IF~ELSE is developed; if P1 = parameter, ELSE~ENDIF is developed.

(3) The LT (less than) operator

[Format]

```
<expression 1> ΔLT Δ<expression 2>
<expression 1> < <expression 2>
```

[Function]

When the value (evaluation) of the lefthand side is less than the value (evaluation) of the righthand side, -1 (true) is returned; when the value (evaluation) of the lefthand side is greater than the value (evaluation) of the righthand side, or is the same, 0 (true) is returned as the result of the operation.

[Explanation]

- o "LT" may be entered as "<".
- o As -1 is processed as twos complements; as a hexadecimal value it is expressed as "0FFFFH".

[Example]

Condition	DAT	02H	] ①
R3	MEM	3.45H	
	:		
Macro	MACRO	P1,P2,P3	] ②
	IF	P1 LT Condition	
	SUB	R1,#P2	
	:		
	ELSE		
	SUB	R1,#P3	
	:		
	ENDIF		
	ENDM		

(Comments)

- ① Defines a symbol.
- ② Carries out a macro definition. P1, P2 and P3 are temporary parameters. If P1 < condition, IF~ELSE is developed; if P1 ≥ condition, ELSE~ENDIF is developed.

(4) The LE (less than or equal) operator

[Format]

```
<expression 1> Δ LE Δ <expression 2>
<expression 1> <=<expression 2>
```

[Function]

When the expression value on the left is smaller than or equal to the value (evaluation) on the right, -1 (true) is returned; and when the value (evaluation) on the left is greater than the value (evaluation) on the right, 0 (false) is returned as the result of the operation.

[Explanation]

- o "LE" may be entered as "<=".
- o As -1 is processed as twos complements, in hexadecimal it is expressed as "0FFFFH".

[Example]

Condition	DAT	04H	] ①
R3	MEM	1.13H	
	:		
Macro	MACRO	P1,P2,P3	] ②
	IF	P1 LE Condition	
	SUB	R1,#P2	
	:		
	ELSE		
	SUB	R1,#P3	
	:		
	ENDIF		
	ENDM		

(Comments)

- ① Defines a symbol.
- ② Effects a macro definition. P1, P2 and P3 are temporary parameters. If P1 < condition, IF~ELSE is developed; if P1 > condition, ELSE~ENDIF is developed.

## UMAS17K ASSEMBLER

(5) The GT (greater than) operator

[Format]

```
<expression 1> Δ GT Δ <expression 2>  
<expression 1> > <expression 2>
```

[Function]

When the expression value on the left is smaller than or equal to the value (evaluation) on the right, -1 (true) is returned; and when the value (evaluation) on the left is greater than the value (evaluation) on the right, 0 (false) is returned as the result of the operation.

[Explanation]

- o "GT" may be entered as ">".
- o As -1 is processed as twos complements, in hexadecimal it is expressed as "0FFFFH".

Example:

Condition	DAT	07H	] ①
R3	MEM	3.44H	
	:		
Macro	MACRO	P1,P2,P3	] ②
	IF	P1 GT condition	
	SUB	R1,#P2	
	:		
	ELSE		
	SUB	R1,#P3	
	ENDIF		
	ENDM		

(Comments)

- ① Defines a symbol.
- ② Effects a macro definition. P1, P2 and P3 are temporary parameters. If P1 > condition, IF~ELSE is developed; if P1 ≤ condition, ELSE~ENDIF is developed.

(6) The GE (greater than or equal) operator

[Format]

```
<expression 1> Δ GE Δ <expression 2>  
<expression 1> >= <expression 2>
```

[Function]

When the expression value on the left is smaller than or equal to the value (evaluation) on the right, -1 (true) is returned; and when the value (evaluation) on the left is greater than the value (evaluation) on the right, 0 (false) is returned as the result of the operation.

[Explanation]:

- o "GE" may be entered as ">=".
- o As -1 is processed as twos complements, in hexadecimal it is expressed as "OFFFH".

[Example]

```
Condition  DAT  OFH      ] ①  
          R3  MEM  1.67H  
          :  
Macro     MACRO P1,P2,P3  
          IF   P1 GE Condition  
          SUB  R1,#P2  
          :  
          ELSE  
          SUB  R1,#P3  
          :  
          ENDIF  
          ENDM      ] ②
```

(Comments)

- ① Defines a symbol.
- ② Effects a macro definition. P1, P2 and P3 are temporary parameters. If P1 ≥ condition, IF~ELSE is developed; if P1 < condition, ELSE~ENDIF is developed.

## UMAS17K ASSEMBLER

### 2.9.6 Shift operators

#### (1) The SHR (shift right) operator

##### [Format]

<expression 1> Δ SHR Δ <expression 2>

##### [Function]

The value (evaluation) of the left of the shift operator is bit-shifted to the right the number of times of the value (evaluation) of the right.

##### [Explanation]

The number of effective operation bits is 16 bits. The result of the shift is the insertion of 0 as the MSB.

##### [Examples]

```
Constant DAT 4578H

Memory1 MEM 0.48H
Memory2 MEM 0.49H
Memory3 MEM 0.4AH
Memory4 MEM 0.4BH
      .
      .
      MOV Memory1, # Constant AND 0FH
      MOV Memory2, # Constant SHR 4 AND 0FH
      MOV Memory3, # Constant SHR 8 AND 0FH
      MOV Memory4, # Constant SHR 0CH AND 0FH
```

##### (Comments)

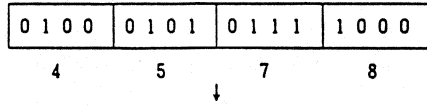
In the example given above, the numeric value which has assigned to the symbol "constant" is placed at bank 0 48H~4BH in the data memory.

" AND 0FH" is, with regard to DAT type expressions, used to indicate that the lower 4 bits only are effective.

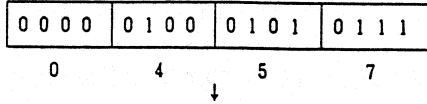
If, in the above example, " AND 0FH" is omitted, an error will be generated.

Below appears an explanation of the processing procedure in the case in which MOV memory1 , # constant SHR 4 AND 0FH.

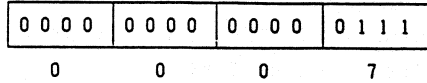
The data 4578H is assigned to "constant".



After executing #constant SHR 4 :



After executing AND 0FH:



0 is inserted as the upper bit.

## UMAS17K ASSEMBLER

### (2) The SHL (shift left) operator

#### [Format]

<expression> Δ SHL Δ <expression 2>

#### [Function]

This operator bit-shifts the value (evaluation) on the left of the shift operator to the left for the number of times of the value (evaluation) of the right of the shift operator.

#### [Explanation]

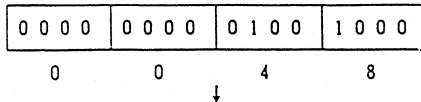
The number of effective bits is 16. The result of the shift is the insertion of 0 as the LSB.

#### [Example]

```
Memory1 MEM 0.48H
      .
      .
      SET1 .FM.Memory1 SHL 4 OR 0001B
      .
      .
```

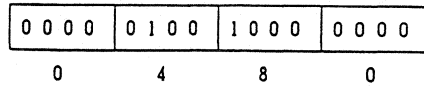
#### (Comments)

In the example given above, a symbol defined as an MEM type symbol is being used, and 1 bit (memory 1 LSB) in the data memory specified by that symbol is being set. SET1 is a built-in macro instruction which sets the position flag entered in the operand, and ".FM." is a type conversion function which converts the memory type symbol to a flag type symbol. The procedure for processing SET1 .FM.memory1 SHL 4 OR 0001B is explained below. 0.48H data is assigned to memory 1. First, memory 1 is type converted from a MEM type symbol to an FLG type symbol. This has no effect on the value 0.48H.

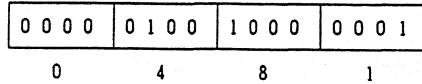




After executing memory 1 SHL 4;



After executing OR 0001B;



0 is inserted as the lower bit.

## UMAS17K ASSEMBLER

---

### 2.9.7 Others

(1) ( ) (parentheses) operation sequence designator

#### [Format]

<expression 1>operator(<expression 2>operator<expression 2>)  
(<expression 1>operator<expression 2>(operator)<expression 2>

#### [Function]

Operations enclosed in parentheses will be executed first, without reference to the order of priority of operators.

#### [Explanation]

- o If parentheses are nested, expressions will be operated on from the parentheses furthest inside.
- o A maximum of 16 levels of nesting is possible.
- o If the maximum number of levels of nesting is exceeded, an S error (stack overflow) will be generated.

#### [Example]

```
Constant1 DAT 4789H
Constant2 DAT 3H
Memory1 MEM 0.48H
      .
      .
      MOV Memory1, #((Constant1+Constant2)*04H)
      .
      .
      .
```

#### (Comments).

Parentheses may be used to designate the order of priority of operations when carrying out operations with a number of operators.

### 2.10 Functions

The functions which may be used with the AS17K are as given below.

#### (1) The type conversion function

This function performs type conversion of symbols.

With the AS17K, the type is defined at the same time as a value is assigned to a symbol. For this reason, it is possible with the AS17K, when generating source programs, to detect automatically entry errors in which the wrong symbol has been entered. In this kind of situation, the symbol type can be changed by using the type conversion function.

#### (2) The location counter function

This function returns the current location counter value.

#### 2.10.1 The type conversion function

Whenever a symbol value is defined with the AS17K, a type is assigned at the same time. There are types which permit entry to operands with mnemonics, and types which do not.

If symbols of types which do not permit entry are to be used in the operand field, a type conversion must be effected with the type conversion function. If symbols of a type which are not permitted are used, an O error (illegal operand type) will be generated.

[Format]

. <type after conversion> <current type>. <symbol>

Type	Description format
Data type	D
Memory type	M
Flag type	F
Label type	L

[Function]

Converts to the type evaluation designated.

## UMAS17K ASSEMBLER

### [Explanation]

- o The upper case letters D, M, F and L are used to express data, memory, flag and label types within a type conversion function.
- o The type conversion function must be enclosed in periods.

### [Example]

MEMORY	MEM	0.38H	
DATA	DAT	.DM.MEMORY AND 7FH	
LABEL	LAB	356H	
FLAG	FLG	.FL.LABEL SHL 4+08H	
	.		
	MOV	MEMORY,#.DL.LABEL AND 0FH	; ①
	MOV	.MF.FLAG SHR 4, #DATA	; ②
	SETI	.FM.MEMORY SHL 4+1	; ③
	MOV	.MD.DATA,#.DL.LABEL AND 0FH	; ④

### (Comments)

In this example, the following conversions take place:

- ① Label type to data type.
- ② Flag type to memory type.
- ③ Memory type to flag type.
- ④ The first operand from data type to memory type.
- ⑤ The second operand from label type to data type.

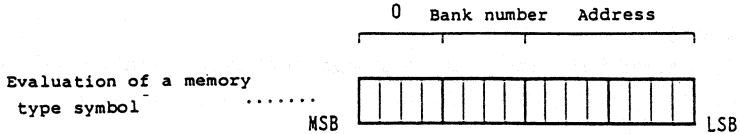
### <Memory and flag values>

Four-nibble values are assigned to memory and label type symbols. The evaluations of these are as given below.

### Memory type symbols:

Memory type symbols are defined by delimiting bank numbers and addresses with bit segment indicators (.). In these values,

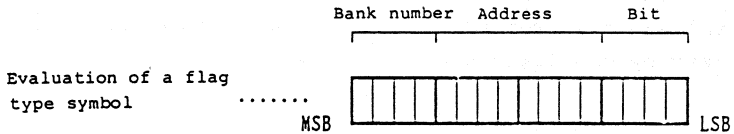
the value of the address below the bit segment indicator is assigned to the second nybble from the bottom, the bank number is assigned to the third nybble from the bottom, and this becomes the valuation of the memory type symbol, as shown in the diagram below.



For example, the evaluation of the memory type symbol defined as 1.23H is 0123H.

### Flag type symbols:

A flag type symbol is defined and delimited by bit segment indicators for the bank number, address and bit position. Of these values, the bank number is assigned to the highest or fourth nybble, while the address remains as it is in the following two nybbles after the bank number. However, the bit value is 1, 2, 4 or 8 to correspond with bit positions 0, 1, 2 or 3. That is to say, the value relates to the bit position.



For example, the evaluation of the flag type symbol defined as 3.08H.2 is 3084.

In the example given above, also, the value of the symbol "MEMORY" is 00038H, and the value of FLAG is 3568H (bank 3 address 56H bit position 3).

## 2.10.2 The location counter function

This function indicates the current location counter value.

## [Format]

\$

## [Function]

This function indicates the current location counter value.

## [Explanation]

It is possible, by using \$, to execute references to relative addresses easily.

## [Example]

Memory	MEM	0.47H
	.	
	ADD	Memory,#01H
	SKT	Memory,#01H
	BR	\$\$-2

## (Comments)

In the example given above, "\$-2" "(the current location counter)"-2 is expressed. It is possible to display relative addresses by combining \$ with another operator.

### 2.11 Variables used when assembling

These variables are entered into a source program; their values are defined when assembling starts, and through this it becomes possible to control assembler operations.

#### 2.11.1 ZZZn

##### [Function]

SET definition directives entered in assemble options and source programs are symbols which can be used to assign values. These symbols are DAT type symbols. Values specified by the assemble option are set when assembling starts up. They are handled in the source program as DAT type symbols, and it is possible by using the SET definition DIRECTIVE to alter values whenever required.

##### [Format]

Designated with the assemble option.

/ZZZn=m

n: 0 to 9

m: desired value

10 values may be set from ZZZ0 to ZZZ9. It is possible to define values in up to 16 bits. Values may be expressed in binary (B), octal (O or Q), decimal (D) or hexadecimal (H). However, if character constants or operation expressions are entered, an invalid option message will be generated, and assembling will halt. If no designation has been made with the option, the default value will be set as 0.

##### [Explanation]

- o DAT type symbols can be used in exactly the same way in source programs.
- o If no designation is made with the option, the initial value will default to 0.
- o A PUBLIC declaration cannot be made with ZZZn.

## UMAS17K ASSEMBLER

- o When programs are split into modules, the values designated by the assemble option will be reset when the assembling of that module begins.

[Example]

Below is an example of a source program using ZZZn.

```
MAC      :
          MACRO X
          IF X
            MOV MEM00 ,#ZZZ7      ; ①
          ELSE
            MOV MEM01 ,#ZZZ7      ; ②
          ENDIF
          :
          ENDM
          :
          MAC ZZZ0                  ; ③
          :
ZZZ0     SET  1H
          MAC ZZZ0                  ; ④
```

The comments relate to the situation in which the program given above is assembled using the sequence file TEST.SEQ illustrated below.

```
; TEST.SEQ FILE
; DEV FILES
  D17000.DEV
; OPTION LIST
  /LIS/ROW=70
  /ZZZ0=0H
  /ZZZ7=0FH
; SOURCE FILE
  TST17000.ASM
```

(Comments)

In this example, the assemble option switches ZZZ0 and ZZZ7 are assigned respectively to 0H and 0FH. With the macro "MAC", the value of the parameter X determines whether ① or ② is developed. At ③, because the parameter ZZZ0 value is 0H as designated by the option, ② is developed. At ④, because the parameter ZZZ0 value is 1H, which was assigned immediately prior, ① is developed.



### 2.11.2 ZZZSKIP

#### [Function]

ZZZSKIP is a variable whose value is set selectively according to the conditions obtaining at assemble time. If the statement generating the object immediately prior to ZZZSKIP is a skip instruction (SKE, SKNE, SKGE, SKLT, SKT, or SKF), or if it is a built-in macro instruction with a skip function (SKTn or SKFn), ZZZSKIP is set to -1(0FFFFH); if this is not the case, it is set to 0.

#### [Example]

```

SKT1 MACRO FLAG
  IF .DF.FLAG AND 800H ; ①
    N3 LAB N2+1
      IF ZZZSKIP
        BR N1
        BR N3
      ENDIF
    N1:
      PEEK WR,.MF.(FL AG SHR4) AND OFH
      SKT WR.#.DF.FLAG AND OFH
    N2:
      ELSE
        SKT .MF.FLAG SHR 4,#.DF.FLAG AND OFH ; ③
      ENDIF
  ENDM
  .
  .
  .
  SKE AA,#2
  SKT1 INTF ;INTF is the flag in the
  BR ABC ; register file. ④
  .
  .
  .
  MOVE BB,#1
  SKT1 INTF
  BR EFG ⑤

```

#### (Comments)

At ①, the flag is set to DAT type, and a check is made of whether or not it is a flag in the register file at 800H. If it is the case that it is a register file flag, ② is developed; otherwise, ③ is developed.

Macro development at ④ is as set out below.

SKE AA,#2	]	Result of developing SKTI INTF.
BR N1		
BR N3		
[N1:] PEEK WR,.MF.(INTF SHR4) AND OFH		
SKT WR,.DF.INTF AND OFH		
[N2:] BR ABC		
[N3:]		

In the case of ⑤:

MOV BB,#1	]	The result of developing SKTI INTF.
PEEK WR,.MF.(INTF SHR4) AND OFH		
SKT WR,.DF.INTF AND OFH		
BR ABC		

**CHAPTER 3 DIRECTIVES AND CONTROL INSTRUCTIONS****3.1 An Outline of Directives and Control Instructions**

The basic function of an assembler is to convert instructions into machine language. The purpose of directives and control instructions is to make the assembler easier to use, and generate lists which are easier to read. Directives and control instructions are not converted to machine language; they instruct the assembler itself. Built-in macro directives, however, are converted into machine language.

Directives are classified as follows:

- (1) The location counter control directive  
ORG
- (2) Symbol definition directives  
DAT, MEM, FLG, LAB  
SET
- (3) Public definition and reference directives  
PUBLIC~BELOW~ENDP  
EXTRN
- (4) Data definition directives  
DW, DB
- (5) Assemble directives with conditions  
IF~ELSE~ENDIF  
CASE~EXIT~OTHER~ENDCASE
- (6) Iteration directives  
REPT~ENDR  
IRP~ENDP  
EXITR
- (7) The macro definition directive  
MACRO~ENDM
- (8) The symbol global declaration directive in macros  
GLOBAL
- (9) The assemble terminate directive  
END

**UMAS17K ASSEMBLER**

---

The control instructions are as follows:

- (1) Output list control instructions
  - TITLE
  - EJECT
  - LIST, NOLIST
  - SFCOND, LFCOND
  - C14344, C4444
- (2) Macro development print control instructions
  - SMAC
  - NOMAC
  - OMAC
  - LMAC
- (3) Source input control instructions
  - INCLUDE
  - EOF
- (4) Documentation generation control instructions
  - SUMMARY
  - ;. (TAG)

### 3.2 Directives

Directives may be entered in the AS17K Mnemonic field. Directives correspond to the individual device being used, and are supplied in a device file. For details, please refer to the device file operating manual for each product. The appendix also contains some information on directives for various devices.

#### 3.2.1 The location counter control directive

---

ORG	ORIGIN	ORG
-----	--------	-----

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	ORG	<expression>	[;comment]

<expression>	≡	<numeric value>
	≡	<numeric value><operator><numeric value>
	≡	<LAB type symbol>
	≡	<LAB type symbol><operator><numeric value>
	≡	<LAB type symbol><operator><LAB type symbol>
	≡	<expression(LAB type)><operator><numeric value>
	≡	<expression(LAB type)><operator><LAB type symbol>

### [Function]

Sets the location counter value.

### [Applications]

- (1) The uPD17000 series program memory is segmented into 8K steps. ORG must be entered at the start of each segment (however, segment 0 (addresses 0000H to 1FFFH) is not needed).
- (2) Designates a table area start address. This ensures that there will be no effect on the table area address if changes are made before the table area address.

### [Explanation]

- (1) Symbols may also be used in the Operand field, but they must previously have been defined as LAB type symbols.
- (2) If no address is designated with the ORG directive at the beginning of the program, the assembler will assign address 0000 to the location counter.
- (3) If the address value designated with the ORG directive is lower than the previous location counter value, an O error (ORG address error) will be generated.
- (4) Type conversion is necessary if differing types are used.
- (5) With labels attached to ORG directives, the immediately prior location counter value is assigned.

[ Example ]

```

MOV     ARO, #.DL.Reference_data AND OFH
MOV     AR1, #.DL.Reference_data SHR 4 AND OFH
MOV     AR2, #.DL.Reference_data SHR 8 AND OFH
MOV     AR3, #.DL.Reference_data SHR 12 AND OFH
ADD     ARO, A
ADDC    ARO, #0

      ORG     700H
Reference_data :
      DW     1234H
      DW     2344H
      DW     5678H
    
```

(Comments)

The above is an example of a reference to a table area. The first address in the table area is defined with the ORG directive, and the MOV instruction is used to store table addresses subsequent to 700H in the address register (AR0~AR3).

3.2.2 Symbol definition directives

Symbol definition directives are instructions which are used to define as desired numeric values, data memory addresses, flags or labels. With the AS17K, symbol types are fixed so as to reduce the incidence of bugs in programming. These instructions are also useful when debugging, when generating documentation and when using the convenient memory map generation function. The AS17K provides the following four types of symbols.

Symbol definition directive name	Type
DAT	Data type (DAT type)
MEM	Data memory address type (MEM type)
FLG	Flag type (FLG type)
LAB	Label type (LAB type)

DAT

DATA

DAT

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
Name	DAT	<expression(DAT type)>	[;comment]
<pre> &lt;expression(DAT type)&gt;≡&lt;numeric expression&gt;                     ≡&lt;DAT type symbol&gt;                     ≡&lt;numeric value&gt;&lt;operator&gt;&lt;numeric value&gt;                     ≡&lt;DAT type symbol&gt;&lt;operator&gt;                       &lt;numeric value&gt;                     ≡&lt;expression(DAT type)&gt;&lt;operator&gt;                       &lt;numeric value&gt;                     ≡&lt;expression(DAT type)&gt;&lt;operator&gt;                       &lt;DAT type symbol&gt;                 </pre>			

### [Function]

This instruction assigns the value of the expression entered as the operand to the name entered in the Symbol field. The name type is data type.

### [Applications]

This directive may be used to give a meaningful name to immediate data which is not numeric data.

### [Explanation]

- (1) Any symbol entered in the Operand field must have already been defined as a DAT type symbol.
- (2) Names must always be delimited by spaces or tabs.
- (3) Since, if an error is made in entering a symbol or a mnemonic, its name will not be registered, statements which reference that name will also generate errors. In the case of an operand entry error, 0 is assigned to the name.
- (4) Names defined with the DAT directive may not be redefined with different values within the same module.
- (5) Type conversion is necessary if using a symbol of a differing type.

[Example]

```
wait_five_minutes  DAT    5                ; ①
wait_ten_minutes   DAT    wait_five_minutes + wait_five_minutes ; ②
                  .
                  .
                  MOV    time_counter, #wait_five_minutes ; ③
                  .
                  .
```

(Comments)

- (1) The immediate data used at ① and ③ has been defined with a DAT directive.
- (2) "wait\_ten\_minutes" is defined using the "wait\_five\_minutes" defined at ① and ②.



MEM

DATA MEMORY ADDRESS

MEM

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
Name	MEM	<expression(MEM type)>	[;comment]
<p>&lt;expression(MEM type)&gt; ≡ &lt;expression(DAT type)&gt;.            &lt;expression(DAT type)&gt;            ≡ &lt;MEM type symbol&gt;            &lt;MEM type symbol&gt;            ≡ &lt;operator&gt;&lt;expression(DAT) type&gt;</p> <p>&lt;For the block &lt;expression(DAT type)&gt;, please refer to the DAT directive. "." is a position segment marker.</p>			

### [Function]

Assigns the value of the expression designated by the operand to the name entered in the Symbol field. The type of that name is a data memory address type.

### [Applications]

This instruction is used in the definition of data memory, register file, and system register addresses.

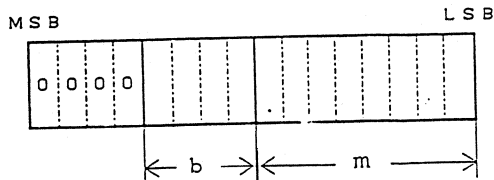
### [Explanation]

- (1) The position segment marker "." must be used. If it is not used, a T error (type error) will be generated. The significance of using this position segment marker is explained below.

b . m

b : bank number

m : data memory address



## UMAS17K ASSEMBLER

- (2) Names must be delimited by spaces or tabs.
- (3) If errors occur when entering symbols or mnemonics, the name will not be registered and statements which make references to that name will generate errors. If an error is made in an operand entry, 0 will be assigned to the name.
- (4) Names defined with the MEM directive may not be redefined within the same module.

### [Examples]

```
; Data memory address definition
Time_position_10th_place MEM 0.10H ; ①
Time_position_1st_place MEM Time_position_10th_place+1 ; ②
.
.
.
ADD Time_position_1st_place,#1 ; ③
ADDC Time_position_10th_place,#0
```

### (Comments)

- ① The 10H address in data memory bank 0 is assigned to the name "time\_position\_10th\_place."
- ② The next address (11H) is assigned to "time\_position\_1st\_place".
- ③ At 3, 1 is added (time\_position\_10th\_place, time\_position\_1st\_place.)

---

FLG

FLAG

FLG

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
Name	FLG	<expression(FLG type)>	[;comment]
<p>&lt;expression(FLG type)&gt;</p> <p>≡ &lt;expression(DAT type)&gt;. &lt;expression(DAT type)&gt; &lt;expression (DAT type)&gt;</p> <p>≡ &lt;expression(MEM type)&gt;. &lt;expression (DAT type)&gt;</p> <p>≡ &lt;FLG type symbol&gt;</p> <p>The block &lt;expression (DAT type)&gt;, refers to the DAT directive; for the &lt;expression (MEM type)&gt; block, please refer to the MEM directive. "." is the position segment marker.</p>			

[Function]

Assigns the value of the expression designated by the operand to the name entered in the Symbol field. The type of the name is FLAG type.

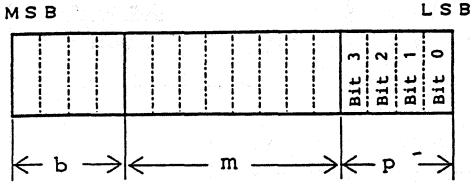
[Applications]

This instruction is used to define 1 bit in the data memory, register file or system register.

[Explanation]

(1) The position segment marker "." must be used. If it is not used, a T error (type error) will be generated. The use of the position segment marker is explained below.

b . m . p                      b: bank number  
    m: data memory address  
    p: bit position (ineffective other than 0, 1, 2, 3,)



Value of p	p' (evaluation of p)	Designated bit position
0	1	Bit 0 (LSB)
1	2	Bit 1
2	4	Bit 2
3	8	Bit 3 (MSB)

- (2) Names must be delimited by spaces or tabs.
- (3) If errors occur when entering symbols or mnemonics, the name will not be registered and statements which make references to that name will generate errors. If an error is made in an operand entry, 0 will be assigned to the name.
- (4) Names defined with the FLG directive may not be redefined within the same module.

[Example]

```

; List of FLAGS
24_hour_display_flag - FLG 0.20H.1 ; ①
.
.
.
; I/O terminal
Buzzer_terminal FLG POA1 ; ②
.
.
.
; A-signed to LCD display
LCD_display_0 MEM 0.60H
AM_display FLG LCD_display 0.3
PM_display FLG LCD_display 0.2
Colon_display FLG LCD_display 0.1
Timer_display FLG LCD_display 0.0 ] ③
.
.
.
SKF1 24_hour_display_flag ] ④
CLR2 AM_display, PM_display
.
.
.

```

(Comments)

At ①, the data memory address desired is selected directly, and 1 bit within it is defined.

At ②, the name (POA1) registered in 2 bits of port A in bank 0 with reserved words is altered to a name ("buzzer\_terminal") which is used in that program.

At ③, the memory desired is defined with an MEM type, and each bit is allocated as an FLG type.

At ④, if the 24\_hour\_display\_flag is 1, the AM\_display and PM\_display flags are set.

---

LAB	LABEL	LAB
-----	-------	-----

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
Name	LAB	<expression(LAB type)>	[;comment]
<pre> &lt;expressionLAB type&gt; ≡ &lt;numeric value&gt;                         ≡ &lt;LAB type symbol&gt;                         ≡ &lt;numeric value&gt;&lt;operator&gt;                           &lt;numeric value&gt;                         ≡ &lt;(LAB type symbol)&gt;&lt;operator&gt;                           &lt;numeric value&gt;                         ≡ &lt;expression (LAB type)&gt;&lt;operator&gt;                           &lt;numeric value&gt;                         ≡ &lt;(LAB type symbol)&gt; &lt;operator&gt;                           &lt;LAB type symbol&gt;                     </pre>			

[Function]

Assigns the value of the expression designated in the Operand field to the name entered in the Symbol field. The type of the name is label type.

[Application]

The LAB directive is used when it is desired to use a different name for a name entered in the label field.

[Explanation]

- (1) Any symbol entered in the Operand field must have already been defined as a LAB type symbol.
- (2) Names must always be delimited by spaces or tabs.
- (3) Since, if an error is made in entering a symbol or a mnemonic, its name will not be registered, statements which reference that name will also generate errors. In the case of an operand entry error, 0 is assigned to the name.
- (4) Names defined with the LAB directive may not be redefined with different values within the same module.

- (5) Type conversion is necessary if using a symbol of a differing type.
- (6) If the value of an expression in the Operand field is not within the range permitted for the program memory address for that product, an R error (ROM address error) will be generated.

### [Examples]

```
Module 1
PUBLIC  Module_1_flag_reset
Flag_reset:
        CLR3  FLAG1,FLAG2,FLAG3
        .
Module_1_flag_reset  LAB  Flag_reset
        .
        .
        RET
```

```
Module 2
EXTRN  LAB: Module_1_flag reset
        .
        .
        CALL Module_1_flag reset
        CALL Flag_reset
        .
        .
        .
Flag_reset:
        CLR2  F1,F2
        .
        .
        RET
```

(Comments)

In this example, module 1 and module 2 have the same `flag_reset` label, and the `module_1_flag_reset` is used in module 2. Normally, when using a label from another module, `PUBLIC` and `EXTERN` declarations are carried out, but in this example, where the name is the same, another name must be amended and used. So, using the `LAB` directive, `flag_reset` in the module 1 is defined as the separate name `module_1.flag_reset`.



SET

SET

SET

<u>Symbol</u> Name	<u>Mnemonic</u> SET	<u>Operand</u> <expression>	<u>Comment</u> [;comment]
<p>&lt;expression&gt; ≡ &lt;numeric value&gt; ≡ &lt;expression (DAT type)&gt; ≡ &lt;expression (MEM type)&gt; ≡ &lt;expression (FLG type)&gt; ≡ &lt;expression (LAB type)&gt; ( &lt;expression&gt; ≡ &lt;numeric value&gt; is DAT type)</p> <p>For more on the above four blocks, please refer to the directive for each symbol.</p>			

### [Function]

Assigns the value of the expression entered in the Operand field to the name entered in the Symbol field. The type of the name and the type of the expression must be the same. However, numeric values with no position segment marker will automatically become data types.

### [Applications]

This instruction is used when setting temporary parameters with the assemble directive with condition attached (IF~LSE~ENDIF, CASE~EXIT~OTHER~ENDCASE) or the iteration directives (REPT~ENDR, IRP~ENDP and EXITR), or if setting variables when assembling such as ZZZn or ZZZSKIP.

### [Explanation]

- (1) Names must be delimited by spaces or tabs.
- (2) If errors occur when entering symbols or mnemonics, the name will not be registered and statements which make references to that name will generate errors. If an error is made in an operand entry, 0 will be assigned to the name.

- (3) The type of the name defined by the SET directive will be the same as the type of the symbols comprising the expression entered in the Operand field.
- (4) Note that the built-in macro instruction SETn is a different directive.
- (5) If a name has been defined with the SET directive, and it is desired to redefine it with the same name, the type cannot be changed, but the detail of a name of the same type can be changed with ease.
- (6) Symbols which are defined by the SET directive will have no effect on memory maps or flag maps.

## [Examples]

```

; If the condition is 1, M0 is 10H in bank 0.
; If the condition is 2, M0 is 20H in bank 0.
; If the condition is 3, M0 is 30H in bank 0.
; In other cases, M0 is 00H in bank 0.
;
MO00 MEM 0.00H
MO10 MEM 0.10H
MO20 MEM 0.20H
MO30 MEM 0.30H
;
;
Condition DAT 1 ; ①
MO SET MO00
IF Condition = 1 ; If the condition
MO SET MO10 is 1.
ENDIF
IF Condition = 2 ; If the condition
MO SET MO20 is 2.
ENDIF
IF Condition = 3 ; If the condition
MO SET MO30 is 3.
ENDIF
.
.
MOV MO,#0

```

## (Comments)

At ①, the condition is set as 1. Therefore, in the above example, M0 becomes the 10H address in bank 0.

### 3.2.3 Public definition and public reference directives

These directives refer to or define symbols which are used jointly by more than one module.

External definition directive

PUBLIC~BELOW~ENDP

External reference directive

EXTRN

```
PUBLIC
BELOW
ENDP
```

```
PUBLIC
BELOW
END PUBLIC
```

```
PUBLIC
BELOW
ENDP
```

Format 1:

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Labels:]	PUBLIC	<symbol group>	[;comment]
<symbol group> ≡ <symbol> ≡ <symbol>, <symbol> ≡ <symbol group>, <symbol>			

Format 2:

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Labels:]	PUBLIC	BELOW	[;comment]
[name	DAT	<expression (DAT type)>	[;comment]
[name	MEM	<expression (MEM type)>	[;comment]
[name	FLG	<expression (FLG type)>	[;comment]
[name	LAB	<expression (LAB type)>	[;comment]
	ENDP		

[Function]

There are two formats for the public definition directive. In the first format, a declaration is made that the symbol entered in the Operand field is referenced in another module. In the second format, a declaration is made that the symbol defined in the block enclosed by "PUBLIC BELOW" and "ENDP" is referenced in another module.

[Application]

Declares that a symbol is referenced in another module.

### [Explanation]

- (1) The public definition directive may be entered at any position in a source program.
- (2) In format 1, it is necessary to define a publicly declared symbol using the symbol definition instruction in the same module. If a symbol for which this has not been done is entered, an S error (undefined symbol) is generated.
- (3) In format 2, the symbol defined may be used as a symbol within a module.

### [Example]

(Example 1)

```
PUBLIC Hour-10th-place, Hour-1st-place, Minute-10th-place,  
Minute-1st-place, Wait-5-minutes ; ①
```

```
Hour-10th-place MEM 0.10H  
Hour-1st-place MEM 0.11H  
Minute-10th-place MEM 0.12H  
Minute-1st-place MEM 0.13H  
Second-10th-place MEM 0.14H
```

```
Wait-5-minutes DAT 5
```

### (Comments)

In example 1, format 1 is used to make a public declaration. At ①, the five symbols "Time\_10th\_place", "Time\_1st\_place", "Minute\_10th\_place", "Minute\_1st\_place" and "Wait\_5\_minutes" are publicly defined.

(Example 2)

```

Hour-10th-place      PUBLIC      BELOW
                     MEM         0.10H
Hour-1st-place       MEM         0.11H
Minute-10th-place   MEM         0.12H
Minute-1st-place    MEM         0.13H
                     ENDP
Second-10th-place   MEM         0.14H
                     .
                     .
                     PUBLIC BELOW
Wait-5-minutes      DAT 5
                     ENDP
    
```

} ①

; ②

(Comments)

Example 2 shows the same detail as is used in example 1 with format 2 applied. The four symbols "Hour\_10th\_place", "Hour\_1st\_place", "Minute\_10th\_place" and "Minute\_1st\_place" are enclosed by "PUBLIC BELOW" and "ENDP" at ① and publicly declared. At ②, there is a public declaration in format 2 of "Wait.5.minutes".

---

EXTRN

EXTERN

EXTRN

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	EXTRN	<expression>: <symbol group>	[;comment]
<expression>	≡ DAT	<symbol group>	≡ <symbol>
	≡ MEM		≡ <symbol>, <symbol>
	≡ FLG		≡ <symbol group> ,
	≡ LAB		<symbol>
However, the types in the <expression> and <symbol group> in the Operand field must agree.			

[Function]

Declares that there is a reference in the current module to a symbol, entered in the Operand field, which has been publicly declared in another module.

[Application]

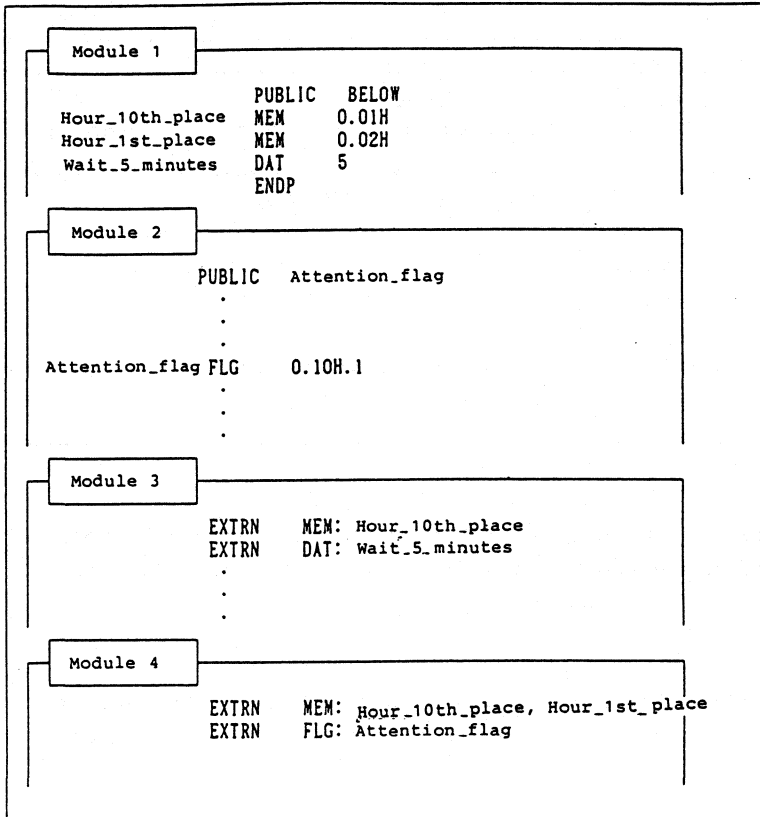
This instruction makes it possible to make use in the current module of a required symbol which is publicly declared in another module.

[Explanation]

- (1) With the EXTRN directive, the symbol which is declared must be entered in that module before the reference is made.
- (2) <expression> and <symbol> are delimited by en columns.
- (3) If a symbol of a type other than that designated in the expression is entered, a T error (invalid type error) will be generated.

## UMAS17K ASSEMBLER

[Example]



(Comments)

In modules 1 and 2, a symbol used in modules 3 and 4 is publicly declared. Of the symbols publicly declared in modules 3 and 4, only the symbol used is EXTRN declared.

### 3.2.4 Data definition directives

The data definition directives define table area data. There are two varieties of data definition directive:

DW: defines 16-bit length data.

DB: defines 8-bit length data.





[Example]

```
MOV ARO,#.DL.Table AND OFH
.
.
.
Table:

DW 1234H
DW '0'
```

(Comments)

In the example given above, "table" is the data area, and each datum is defined with the DW directive.

DB

DEFINE BYTE

DB

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	DB	<8-bit data group>	[;comment]
<p>&lt;8-bit data group&gt; ≡ &lt;expression(DAT type)&gt;            ≡ '&lt;character string&gt;'            ≡ &lt;8-bit data group&gt;,            &lt;expression (DAT type)&gt;            ≡ &lt;8-bit data group&gt;,            '&lt;character string&gt;'</p>			

[Functions]

Assigns the characters or expression entered in the Operand field to the current location counter value (program memory address) as an 8-bit object code.

[Application]

Used in defining table area data.

[Explanation]

- (1) It is possible to designate up to 32 individual operands delimited by commas. If more than this number is entered, a C error (operand count error) is generated.
- (2) Character constants enclosed in quotation marks may be converted to 8-bit ASCII codes and up to 32 characters entered.
- (3) Where codes are 8-bit only, 0 will appear in the lower 8 bits.  
 DB 'A' → object code : 4100H

**UMAS17K ASSEMBLER**

---

[Examples]

```
DB 01AH
DB 'A'

DB 'ABCDEFGHIJKLMN'
DB OFFH,OFFH OFFH
```

### 3.2.5 Assemble directives with conditions

The effective use of assemble directives with conditions can lead to more efficient programming, and the development of a source program library. There are two types of assemble directives with conditions:

IF ~ ELSE ~ ENDIF

CASE ~ EXIT ~ OTHER ~ ENDCASE

```
IF
ELSE
ENDIF
```

```
IF
ELSE
ENDIF
```

```
IF
ELSE
ENDIF
```

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	IF	<Expression (DAT type)>	[:comment]
[	Statement		]
	[ELSE]		[:comment]
[	Statement		]
	ENDIF		[:comment]

### [Functions]

#### (1) IF and ENDIF

If the evaluation of an Operand field IF statement is a value other than 0 (false), the statement enclosed by IF and ENDIF will be assembled.

If the evaluation of the IF statement in the Operand field is 0 (false), the statement enclosed by IF and ENDIF will not be assembled.

#### (2) IF and ELSE and ENDIF

If the evaluation of an Operand field IF statement is a value other than 0 (false), the statement enclosed by IF and ELSE will be assembled, but the statement enclosed by ELSE and ENDIF will not be assembled. If the evaluation of the Operand field IF statement is 0 (false), the statement enclosed by IF and ELSE will not be assembled, but the statement enclosed by ELSE and ENDIF will be assembled.

### [Application]

This instruction is used to select a statement which is developed in accordance with some routine in a program, and its conditions of use.

## [Explanation]

- (1) All statements enclosed between an IF and its corresponding ENDIF are defined as the IF~ENDIF block.
- (2) Since ELSE is an option, it is not always necessary to designate it. However, if it is designated, it may only be used once in an IF~ENDIF block. If ELSE is designated more than once, an S error (syntax error) will be generated.
- (3) Any symbol entered in an Operand field IF statement must have been previously defined.
- (4) Up to 40 levels of nesting are possible, including built-in macro instructions, macro reference statements, and REPT, IRP and CASE statements.
- (5) It is not possible to enter labels in ELSE and ENDIF statements.

## [Examples]

## (Example 1)

```

; List of conditional items. ; ①
    12_hour_display DAT 0
    24_hour_display DAT 1
    .
    .
; Condition setting table ; ②
    Time_display_conditions DAT 12_hour_display
    .
    .
; Setting the time displayed first with the time ; ③
IF Time_display_conditions
; 12 hour display
    MOV Hour_10th_place ,#1
    .
    .
ELSE
; Performs 24 hour display
    MOV Hour_10th_place ,#0
    .
    .
ENDIF
    .
    .

```

[Comments]

In the above example, a time display is selected using the IF directive.

At ①, the condition items used at ② are listed.

At ②, the desired object is selected from the list at ①.

At ③, what is required is developed in accordance with the condition items set at ②.

For example, statements within IF~ELSE are developed.

[Example]

(Example 2)

```
ADB  MACRO  A,B
      IF (.DM.A SHR 8)=(.DM.B SHR 8)
          ADD  A,B
      ELSE
          PEEK  WR,BANK
          MOV  BANK,#.DM.B SHR 8
          ADD  A,B
          POKE  BANK,WR
      ENDIF
      ENDM
      .
      MOO  MEM  0.10H
      MOI  MEM  0.11H
      DATA  DAT  03H
      .
      ADB  MOO,DATA ; ②
      .
      ADB  MOI,DATA ; ③
```

(Comments)

ADB at ① is a macro which adds together a register A and a memory B. Whatever bank B is, automatic bank switching is effected with the directive.

At ②, the addition of the same bank memory is displayed.

At ③, the addition of a different bank memory is displayed.

CASE  
EXIT  
OTHER  
ENDCASE

CASE  
EXIT  
OTHER  
ENDCASE

CASE  
EXIT  
OTHER  
ENDCASE

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[label:]	CASE	<expression (DAT type)>	[;comment]
		<numeric value>:	[;comment]
		[ Statement ]	
		[EXIT]	[;comment]
		<numeric value>	[;comment]
		[ Statement ]	
		[EXIT]	[;comment]
		.	
		.	
		.	
[OTHER:]		[ Statement ]	
		ENDCASE	[;comment]

[Functions]

This instruction assembles the statement which is enclosed between the numeric value label of the evaluation of <expression (DAT type)> entered in the CASE statement in the Operand field, and ENDCASE. If there is an EXIT occurring in the process, the assemble operation will bypass what is between the following statement and ENDCASE. If there is no numeric value label corresponding to the value of <expression (DAT type)>, the statement between the OTHER and ENDCASE statements will be assembled. However, OTHER must be entered at the end of the numeric value label.



### [Application]

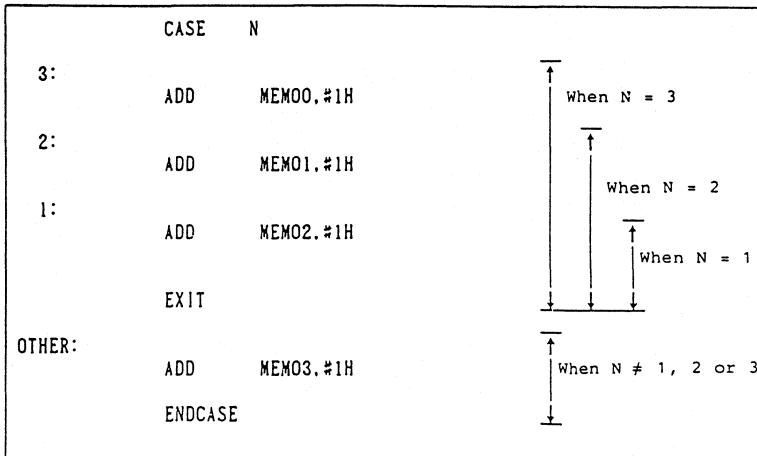
This is used to select the statement to be developed in accordance with the conditions of use in a routine in a program.

### [Explanation]

- (1) Numeric value labels or comments only can be entered in the line containing the numeric value label. The numeric value label is a integer value up to  $1 \leq X \leq 65535$ . If the same numeric value label is entered in the same block twice or more, the numeric value label entered last will be the effective one.
- (2) All statements included between CASE and its corresponding ENDCASE are defined as the CASE- ENDCASE block.
- (3) Up to 40 levels of nesting are possible, including the nesting of built-in macro instructions, macro reference statements, and IF, IRP and REP statements.
- (4) Numeric value labels may be entered in whatever order is desired.
- (5) If a numeric value label is entered at the end of an OTHER statement, an S error (syntax error) will be generated.

### [Example]

(Example 1)



(Comments)

The above four programs are developed from a CASE statement N.

[Example]

(Example 2)

```

CASE   ZZZO           ; ①
0:
    MOV   M1,#0
    MOV   M2,#0
    EXIT

1:
    MOV   M1,#1
    MOV   M2,#1
    EXIT           ] ③

2:
    MOV   M1,#2
    MOV   M2,#2
    ENDCASE
    
```

A>AS17K D17000.DEV /ZZZO=1/ TEST.ASM

②

(Comments)

In the above example, the program makes use of the assemble variable ZZZO CASE directive.

At ①, ZZZO is entered in the CASE Operand field by designating the value ZZZO when assembling, and the corresponding section is developed. ② is an example of what is entered when starting up the assembler. By designating the assemble option "ZZZO=1", 1 is registered at ZZZO in the program. In the example, the point ③ is developed.

### 3.2.6 Iteration directives

The effective use of iteration directives can make programming more efficient. There are two types of iteration directives:

REPT ~ (EXITR) ~ ENDR

IRP ~ (EXITR) ~ ENDR

REPT  
ENDR

REPEAT  
ENDREPEAT

REPT  
ENDR

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	REPT	<expression (DAT type)>	[;comment]
[	Statement		]
	[EXITR]		[;comment]
[	Statement		]
	ENDR		[;comment]

### [Function]

The statement enclosed by REPT and ENDR is developed repeatedly, the number of repetitions being the value of the evaluation of the <expression (DAT type)>.

If EXITR occurs between an REPT and an ENDR, development will terminate, and assembling will continue from the next statement after ENDR.

### [Application]

This instruction is used to repeat the same statement.

### [Explanation]

- (1) Up to 40 levels of nesting are possible, including nesting of built-in macro instructions, macro reference statements, and IF, IRP and CASE statements.
- (2) Symbols used in the <expression (DAT type)> must have been previously defined.

[Example]

```

Table_end  LAB  7FFH                ; ①
           .
           .
Table :    .
           DW  1111H                ; ②
           DW  1112H
           .
           .
Unused table:
           REPT .DL.(Table_end - unused_table)
           DW  0000H                ] ③
           ENDR
    
```

(Comments)

Above is a program which assigns 000H to an unused table area. At ①, first the symbol used at ③ is defined. If it is defined after REPT ENDR, an error will be generated. ② indicates the beginning of the table. At ③, using the REPT directive, 0000H is assigned to the unused table.

IRP  
ENDR

INDEFINITE REPEAT  
END REPEAT

IRP  
ENDR

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	IRP	<Temporary Parameter>, <Actual Parameter Group>	[;comment]
[Statement			]
	[EXITR]		[;comment]
[Statement			]
	ENDR		[;comment]

<Actual parameter group> ≡ <(expression (DAT type))>
≡ <(expression (LAB type))>
≡ '<character string>'
≡ <permanent parameter group>, <expression(DAT type)>
≡ <permanent parameter group>, <expression (LAB type)>
≡ <permanent parameter group>, '<character string>'

### [Function]

This instruction replaces all temporary parameters in a statement enclosed by IRP and ENDR with permanent parameter groups in order from the left, and develops the data number of the permanent parameter groups in the statement enclosed by IR and ENDR. If EXITR appears between IRP and ENDR, development will halt, and assembling will continue from the next statement after ENDR.

### [Application]

This is used when carrying out repeated definitions of similar statements.

## UMAS17K ASSEMBLER

### [Explanation]

- (1) Up to 40 levels of nesting are possible including the nesting of built-in macro instructions, macro reference statements, and IF, REP and CASE statements.
- (2) The entry format for permanent parameter groups is the same as the entry format for permanent parameter groups in macro references.
- (3) If no permanent parameter group is entered, a NULL code is presented.
- (4) Only one temporary parameter may be entered.
- (5) The entry format for the temporary parameter is the same as that for symbols.

### [Example]

```
IRP  DATA,0,1,2
SKNE AAA,&#DATA
OR   A1,#(1 SHL DATA)
SKNE BBB,#(DATA+1)
OR   B1,#( 1000B SHR DATA)
ENDR
```

### [Comments]

The above example is developed as follows:

```
SKNE  AAA,0
OR    A1,1           First time
SKNE  BBB,1
OR    B1,8

SKNE  AAA,1
OR    A1,2           Second time
SKNE  BBB,2
OR    B1,4

SKNE  AAA,2
OR    A1,4           Third time
SKNE  BBB,3
OR    B1,2
```

In this example, the value of memory A1 and B1 takes DATA OR and is stored at A1 and B1. In the first case, 0 is substituted for DATA and developed. In the second case, 1 is substituted for DATA and developed. In the third case, 2 is substituted for DATA and developed.

---

EXITR

EXIT REPEAT

EXITR

---

Symbol

Mnemonic

Operand

Comment

EXITR

[:comment]

---

[Function]

If EXITR appears in a REPT or IRP statement, developments halts, and assembling continues from the next statement after ENDR.

[Application]

This instruction is used in debugging, when it is desired to halt temporarily or prohibit the use of iteration directives.

[Explanation]

- (1) This instruction is only effective in REPT~ENDR and IRP~ENDR instructions.
- (2) If EXITR is entered in other than one of the two blocks mentioned above, a P error (invalid EXITR statement) will be generated.

[Example]

Table end	LAB	7FFH
	.	
	.	
Table:	DW	1111H
	DW	1112H
	.	
	.	
Unused table:		
	REPT	.DL.(Table_end - unused_table)
	EXITR	
	DW	0000H
	ENDR	



(Comments)

The above example shows the halting of processing between REPT and ENDR.

### 3.2.7 Macro definition directives

If the same routine is to be used a number of times in a program, it is generally possible to use procedures which employ subroutines to reduce the number of programming steps. In cases where parameters are different, in processing routines which closely resemble each other but for which subroutines cannot be used, macro functions can be used to increase programming efficiency. The macro definition directive is used when defining such macros. For more details please refer to Section 3.4 on Macro Functions.

MACRO  
ENDM

MACRO  
END MACRO

MACRO  
ENDM

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
Name	MACRO	<temporary parameter group>	[;comment]
	Statement	(macro body)	
	ENDM		

### [Function]

This assigns a macro name to a series of statements (the macro body) between MACRO and ENDM. The name is used as the defined name when making MACRO references.

### [Application]

This is used when macro definitions.

### [Explanation]

#### (1) Macro body

Macro bodies are composed of macro statements, and include symbols, instructions, and directives, MACRO ENDM and comments excepted.

#### (2) Temporary parameter groups

Up to 16 temporary parameters may be entered, delimited by commas.

Temporary parameters are only effective within macro bodies. Permanent parameters are substituted for temporary parameters entered in macro bodies when that macro is referred to.

[Example]

```
      BADD  MACRO  AH,AL,BH,BL
                ADD   AL,BL
                ADDC  AH,BH
                ENDM
                .
                .
                .
      BADD  YH,YL,ZH,ZL
```

] ①

; ②

(Comment)

The above example shows a macro which adds 2 nybbles of memory. (1) is the macro definition section, and (2) the macro reference section.

### 3.2.8 Symbol global declaration directives in macros

The symbol global declaration directive in a macro is known as a GLOBAL directive.

GLOBAL

GLOBAL

GLOBAL

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	GLOBAL	<symbol group>	[;comment]
<p>&lt;Symbol group&gt; ≡ &lt;symbol&gt;                      ≡ &lt;symbol&gt;, &lt;symbol&gt;                      ≡ &lt;symbol group&gt;, &lt;symbol&gt;</p>			

[Function]

Declares symbols used in macros as symbols which may be referenced outside the macro.

[Application]

The GLOBAL directive is used when employing symbols which appear within a macro outside that macro.

[Explanation]

- (1) The GLOBAL directive may only be used within a macro definition (a block enclosed by MACRO and ENDM). If an attempt is made to use it otherwise, a P error (invalid pseudo) will be generated.
- (2) A global declaration must be entered before defining the symbol it applies to. If it is entered after, an S error (symbol multi defined) will be generated.
- (3) The effective range of a symbol which is globally declared is limited to the same source module program.

[Example]

```
AAA  MACRO  A1
      GLOBAL ABC
      ABC  SET  A1
      .
      .
      ENDM
      AAA  0.10H
      MOV  ABC,#1
      AAA  0.20H
      MOV  ABC,#2
```

①

②

③

(Comment)

The example above shows the use of the GLOBAL directive. At ①, the symbol ABC within the macro definition AAA is defined with the SET directive, and it is declared with the GLOBAL directive so that it may be used outside the macro. At ②, "0.10H" is set as the permanent parameter. Thus, the symbol ABC is defined at bank 0 address 10H by means of the SET directive at ①. At ③, the symbol ABC is redefined at bank 0 address 20H so that it can refer to macro AAA.

### 3.2.9 Assemble terminate directives

This indicates the end of a source (program) module.

END

END

END

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	END		

[Function]

Indicates to the assembler the end of a source (program) module.

[Application]

Entered in the final line of a source (program) module.

[Explanation]

- (1) An error will be generated if a line feed (0AH) does not appear at the end of an END directive. Please note that when using the screen editor, storage is possible without the line feed.
- (2) If anything other than a carriage return or a line feed is entered at the end of an END instruction, a warning (statement after END) will be generated, and the statement will be ignored.

[Example]

```

.
.
.
.
END
    
```

(Comment)

The above shows an END directive appearing in the final line of a source program module.

### 3.3 Control Instructions

Control instructions may be entered in the Mnemonic field of the AS17K. These instructions control the printing of macro developments, source input, documentation generation, and output list formats after assembling is finished but before conversion to machine language.

When control instructions and similar functions have been defined in assemble option\*, the assemble option takes priority. The assemble options and their related control instructions are given below. It is possible to designate which instructions are to be effective or noneffective in the assemble option.

Control instruction	Assemble option in which the control instruction is effective.	Assemble option in which the control instruction is noneffective.
LIST	LIST	NOLIST
NOLIST	LIST	NOLIST
SFCOND	NOCOND	COND
LFCOND	NOCOND	COND
SMAC	NOGEN	GEN
NOMAC	NOGEN	GEN
OMAC	NOGEN	GEN
LMAC	NOGEN	GEN

All control instructions are only effective within the same module.

\* Assemble options control whether or not lists are output when assembling takes place. For further details, please refer to Part 2, Section 4.5 on the Assemble Options.

**UMAS17K ASSEMBLER**

## 3.3.1 Output list control instructions

There are eight types of output list control instructions, as shown below; they are used to generate easy-to-read assemble lists.

Output list control instruction	Function	Default value *
TITLE	Prints the title of an assemble list.	—
EJECT	New page	—
LIST NOLIST	Switches assemble lists output on or off	LIST
SFCOND LFCOND	Turns on and off output of assemble lists false condition blocks in assemble instruction statements with conditions.	LFCOND
C14344 C4444	Designates an expression format for an assemble list object.	C14344

\* If default values are not specifically set, the value will be that set when there are no conditions.



---

TITLE	TITLE	TITLE
-------	-------	-------

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	TITLE	'Character string'	[:comment]

[Function]

Inserts a page break in the assemble list, and prints the character string in the Operand field as the title of the list.

[Application]

This is used to print titles on assemble lists, making the lists easier to understand.

[Explanation]

- (1) A maximum of 87 characters may be entered in the character string. If 88 or more characters are entered, an I error (invalid data length) will be generated.
- (2) If the TITLE control instruction appears, the assembler will insert a page break, and print out the title designated in the header. However, if the TITLE control instruction occurs in one line, a new page will not be generated. Further, the TITLE control instruction will itself be printed in the first line after a page break is generated.

[Example]

Source program list

·	
·	
·	
TITLE	'SUBROUTINE'
·	
·	
·	

EJECT

EJECT

EJECT

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	EJECT		[;comment]

[Function]

Creates a page break in an assemble list.

[Application]

Used to force a page break in a routine. The page break allows the assemble list to be more easily read.

[Explanation]

- (1) If the EJECT control instruction occurs, the assembler will insert a page break.
- (2) The EJECT control instruction character string itself will be printed on the page prior to the page break.

[Example]

Source program list.

```

      .
      .
      BR   ABC

DEF:                                EJECT
      .
      .
    
```

---

LIST

LIST

LIST

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	LIST		[;comment]

[Function]

Indicates where output of an assemble list is to start from.

[Application]

Assemble list output may be inhibited where it is not necessary (for example, for definition sections or routines which have been debugged) by using the NOLIST control instruction; the LIST control instruction is used when that instruction is to be negated.

[Explanation]

- (1) When LIST is designated with an assemble option, at the start of each module list control instruction execute status is invoked, and a list is output. This is used to negate the effect of the NOLIST control instruction.
- (2) When NOLIST has been designated by an assemble option, an assemble list will not be output even though the LIST control instruction appears in a source program.

[Example]

Source program list

```
                NOLIST
:
: Data memory definition section
:
MEMORY1  MEM  0.00H
MEMORY2  MEM  0.01H
:
:
START:    LIST                ; Program start
:
:
```

---

NOLIST

NOLIST

NOLIST

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	NOLIST		[;comment]

[Function]

The NOLIST control instruction halts the output of an assemble list.

[Application]

Used to suppress the listing of sections which are not required when outputting an assemble list, for example routines for which debugging is complete, or definition sections.

[Explanation]

- (1) The NOLIST control instruction is itself printed out.
- (2) The number of lines in the sections which are not printed because of the use of the NOLIST control instruction will be counted.
- (3) The NOLIST control instruction is only effective within the one module.
- (4) The NOLIST control instruction takes precedence over other output list control instructions.

[Example]

Source program list

```

LIST
;
;      Flag definition
;
      FLAG1  FLG  0.30H.1
      FLAG2  FLG  0.30H.2
      FLAG3  FLG  0.30H.4
      .
      .
      .
NOLIST
;
;      Data memory definition
;
      MEMORY1 MEM  0.00H
      MEMORY2 MEM  0.01H
      .
      .
      .
LIST
START:          ; Program start
               BR  ABC
NOLIST
;
;      SUBROUTINE
;
      .
      .
      .

```

---

SFCOND      SHORT FORM CONDITION      SFCOND

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	SFCOND		[;comment]

[Function]

Once the SFCOND control instruction is executed, a statement which is skipped because of the application of a directive which judges a condition plus the condition, will not be output to the assemble list.

[Application]

This is used to suppress output of list sections which are not assembled by assemble with conditions instruction statements. This makes the assemble list easier to read.

[Explanation]

- (1) The SFCOND control instruction is only effective when designated with the assemble option NOCOND.
- (2) The SFCOND control instruction is only effective within the one module.

[Example]

Source program list

AAAA	SFCOND		
SET	OFFH		
IF	AAAA		
MOV	A,#5H	←	Assembled
ELSE			
MOV	A,#6H	←	Not assembled, and list not output
ENDIF			

(Comments)

In this example, a list is output from SFCOND to MOV A, #5H, but it is not output between ELSE and ENDIF where the condition is not true. However, "ENDIF" is output to the list.



---

LFCOND      LONG FORM CONDITION      LFCOND

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	LFCOND		[;comment]

[Function]

Once the LFCOND control instruction is set, a section which is skipped because of the application of an assemble with conditions directive condition, will be output to the assemble list.

[Application]

This is used when it is desired to include sections skipped because of the application of an assemble with conditions directive condition in an assemble list.

[Explanation]

- (1) The LFCOND control instruction is effective when designated with the assemble option NOCOND.
- (2) The LFCOND instruction is only effective within the one module.
- (3) The LFCOND instruction is itself output to the assemble list.

[Example]

Source program list

```
SFCOND
    IF A EQ B
    MOV A, #5H
    ELSE
    MOV A, #6H
    ENDF
                                     ] ①

LFCOND
    IF C NE D
    ADD C, #3H
    ELSE
    ADD C, #5H
    ENDF
                                     ] ②
```

## UMAS17K ASSEMBLER

---

(Comments)

At ①, the SFCOND control instruction is effective, so a list is output for what is between either IF ~ ELSE or ELSE ~ ENDIF. The LFCOND control instruction becomes operative from Section ②, and because of this the whole statement is output.

---

C14344    CODE1-4-3-4-4    BIT FORM    C14344

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[label:]	C14344		[;comment]

[Function]

Formats in the object column in an assemble list are expressed hexadecimally as 16-bit object codes in the form 1-bit - 4-bits - 3-bits - 4-bits - 4-bits from the most significant bit after this instruction.

[Application]

This is used after execution of the C4444 instruction, when it is desired to output in C14344 format.

[Explanation]

- (1) When neither the C14344 or C4444 control instructions are designated, the C14344 instruction is effective. It designates the object code output format for assemble lists.
- (2) Data defined by the DW and DB directives may be output in C4444 format (4-bits - 4-bits - 4-bits - 4-bits), which is unrelated to the C14344 control instruction.

[Example]

Source program list

C14344			
MOV	MEM,#3	] ①	
ADD	MEM,#6		
BR	ABCD		
TABLE:			
DW	1234H	] ②	
DW	5678H		
DW	9ABCH		

(Comments)

Section ① is output in 1-4-3-4-4 bit format, similar to the assemble list mentioned above, by means of the C14344 control instruction. However, since the DW and DB directives bear no relation to the C14344 control instruction, the lower part of the assemble list is output in 4-4-4-4-bit format.

C 4 4 4 4

CODE 4-4-4-4 BIT FORM

C 4 4 4 4

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	C4444		[;comment]

[Function]

Object column formats which appear in assemble lists after the incidence of a C4444 control instruction are expressed hexadecimally as 16-bit object codes of the form 4-bits - 4-bits - 4-bits - 4-bits from the most significant bit.

[Application]

This is used when it is desired to express the object column of an assemble list hexadecimally without distinguishing between operation codes and operands.

[Explanation]

- (1) If neither the C14344 or the C4444 control instruction is designated, C14344 will be the effective control instruction. C4444 designates the object code output format of an assemble list.
- (2) The C4444 control instruction is only effective within the same module.

[Example]

Source program list

C14344		
	ADD	REG,#3
	MOV	MEM,@REG
C4444		
	ADD	REG,#3
	MOV	MEM,@REG

### 3.3.2 Macro development print control instructions

These instructions control whether or not, in developing macro statements or iteration directives when assembling, the details of the development are to be output to an assemble list. There are four types of macro development print control instructions, as given below.

Control instruction	Function
SMAC	Outputs an object code only.
NOMAC	Outputs neither object codes nor statements.
OMAC	Outputs object codes and statements for sections where object codes are generated.
LMAC	Outputs all object codes and statements.

These instructions are effective only when designated with the NOGEN assemble option. NOGEN is the assemble option for outputting assemble lists in respect of macro and iteration directives in accordance with the control instructions listed above.

---

SMAC      SHORT FORM MACRO LISTING      SMAC

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	SMAC		[;comment]

[Function]

Suppresses assemble list output of all statements in macro and iteration directive statement development sections.

[Application]

Used when it is desired to suppress printing of macro development sections, and print out the actual object developed only.

[Explanation]

- (1) Object codes only are developed in assemble lists. These are output side by side in groups of 8 instructions.
- (2) The SMAC control instruction is only effective when designated with the assemble option NOGEN. If this is not designated, the default value will be the GEN setting, and the SMAC instruction will be ineffective.

[Example]

Source program list

```
ABC  MACRO
      NOP
      NOP
      NOP
      ENDM
      .
      .
      SMAC
      ABC
      .
      .
```

NOMAC

NO MACRO LISTING

NOMAC

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	NOMAC		[;comment]

[Function]

Suppresses assemble list output of all statements in macro and iteration directive statement development sections.

[Application]

This is used when it is desired to print out macro names only.

[Explanation]

This differs from the SMAC control instruction in that it does not print out object codes. The NOMAC control instruction is only effective when set with the assemble option NOGEN.

[Example]

Source program list

```

CCC  MACRO
      NOP
      NOP
      NOP
      ENDM
      .
      .
      NOMAC
      CCC
      .
    
```



---

### OMAC OBJECT ORIENTED MACRO LISTING OMAC

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	OMAC		[;comment]

#### [Function]

Outputs to an assemble list, from macro and iteration directive statement development sections, statements which generate object codes, but does not output statements which do not.

#### [Application]

Used when it is desired to print out objects and statements from sections which develop objects only.

#### [Explanation]

The OMAC control instruction is only effective when designated with the NOGEN assemble option.

#### [Example]

Source program list.

```
AAA MACRO
BBB SET      OFFH
IF          AAA
  MOV      A,#5H
ELSE
  MOV      A,#6H
ENDIF
NOP
ENDM
:
OMAC
AAA
:
LMAC
AAA
```

---

LMAC    LONG FORM MACRO LISTING    LMAC

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	LMAC		[;comment]

[Function]

This is used to output to a list all statements in macro and iteration directive statement development sections.

[Application]

Used when it is desired to print out all objects and statements in macro and iteration directive statements.

[Explanation]

- (1) If a list output control instruction (NOLIST, LIST) is present in a statement in a developed macro, list output control is effected in accordance with that designation.
- (2) The LMAC control instruction is only effective when designated with the assemble option NOGEN.
- (3) The LMAC directive is always effective at the beginning of all source module assembling operations.

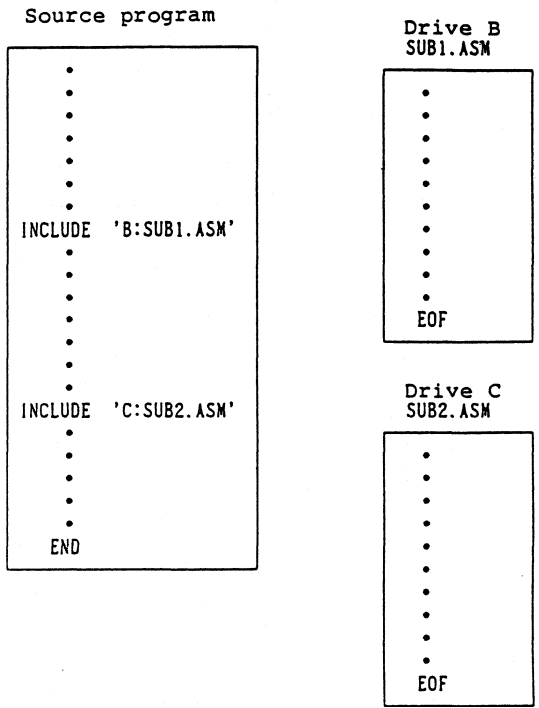
[Example]

Program source list.

```
AAA  MACRO
A    MEM    0.01H
BBB  SET    OFFH
      IF    BBB
          MOV    A,#5H
      ELSE
          MOV    A,#6H
      ENDIF
      NOP
      ENDM
      .
      .
      LMAC
      AAA
      .
      .
```

3.3.3 Source input control instructions

Source input control instructions are used when it is desired to split files in a single program or source module; that is to say, when it is desired to split files because they have become too large, or use programs which have already been completed and placed in a library. The main source input control instruction is INCLUDE. When referencing a file, the INCLUDE control instruction is used to designate the name of that file.



---

INCLUDE

INCLUDE

INCLUDE

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
	INCLUDE	'file name'	[:comment]

Please refer to the Introduction for file naming restrictions.

[Function]

Used to read out source programs designated by the file name, and partially process those source programs.

[Application]

Used when it is desired to insert other split files.

[Explanation]

- (1) A source module designated by INCLUDE may also contain an INCLUDE statement. Eight levels of nesting are possible with INCLUDE. If nine levels or more are set, an N error (INCLUDE nesting error) occurs and those levels are ignored.
- (2) An EOF statement must be placed at the end of the file designated by an INCLUDE control instruction.
- (3) If an extension to the filename is not designated, the extension ASM will be used.
- (4) Since files joined by the INCLUDE control instruction are not in split modules, symbols in the original source program may be referenced as is.

## UMAS17K ASSEMBLER

[Example]

Source module A

```
INCLUDE 'A: MACROFILE.ASM'  
.  
.  
.  
.  
END
```

Source module B

```
INCLUDE 'A: MACROFILE.ASM'  
.  
.  
.  
.  
END
```

MACROFILE.ASM

```
MAC1 MACRO A1,A2  
.  
.  
ENDM  
MAC2 MACRO B1,B2  
.  
.  
ENDM  
MAC3 MACRO C1,C2  
.  
.  
ENDM  
MAC4 MACRO D1,D2  
.  
.  
ENDM  
.  
.  
EOF
```

(Comments)

Only macros used in a number of modules can be put in the one file. If that file is inserted into modules using the INCLUDE control instruction, it is both possible and convenient to use a macro common to a number of source modules without using the PUBLIC or EXTRN directives. If the PUBLIC and EXTRN directives are used, it is necessary to declare the macro name used for each module it is used in.

EOF

END OF FILE

EOF

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
	EOF		

[Function]

This indicates the end of a source file designated by the INCLUDE control instruction. At this point, assembling moves to the next statement after the INCLUDE statement.

[Application]

EOF is used when it is desired to end the file referenced in the INCLUDE control instruction.

[Explanation]

- (1) An error will be generated if the EOF control instruction is not followed by a line feed.
- (2) If a statement is entered after the EOF control instruction, a warning (statement after EOF) will be generated and that statement will be disregarded.

[Example]

Source program

```

.
.
.
.
INCLUDE 'A :SUB1.ASM'
.
.
.
.
END
    
```

Drive A  
SUB1.ASM

```

.
.
.
.
.
EOF
    
```



### 3.3.4 Document generation control instructions

The AS17K's documentation generation functions can be used to output assemble lists and other documentation. Summaries which are entered with the documentation generation control instruction can be extracted and information on symbols, object positions, and assemble lists used in that program can be output in these summaries.

This documentation consists of two sections: the document itself plus the table of contents.

The documentation generation control instructions are SUMMARY and TAG.

SUMMARY

SUMMARY

SUMMARY

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	Summary	'Terminal character string'	[:comment]
		['Title']	

Text not including terminal character string
--

Terminal character string	CR/LF
---------------------------	-------

[Function]

A document may be created by outputting the block from the line after the SUMMARY control instruction to the terminal character string designated by the first operand in SUMMARY. If a heading is designated as the second operand, that heading will be inserted as the table of contents of the document. If an empty character string is designated as the terminal character string, only the heading will be registered, and it will not be possible to enter a character string (text) in the line after the SUMMARY control instruction.

[Application]

Used for outputting text summaries.

[Explanation]

- (1) Carriage returns and line feeds are interchangeable in the document terminal character string.
- (2) There are no limitations on the alphanumeric characters which may be entered in the character string, but the first 16 characters only (8-bit JIS code character conversion) are effective.

- (3) There are no limitations on the alphanumeric characters which may be entered in the title, but only the first 255 characters (8-bit JIS code character conversion) from the beginning are effective, and any more will be ignored.
- (4) The first SUMMARY control instruction to appear in a module will be interpreted as entering the module summary for that module. The second such instruction appearing subsequently will be interpreted as entering a routine summary between that instruction and the next SUMMARY or END directive.
- (5) A line in the summary character string up to the terminal character string in which the first character is a period will be interpreted as a command line for the document. The commands are as given below.

- . EJECT..... Inserts a page break
- . SPnn..... Indicates that a number of spaces is to be output before the data list line which is output continuously and carries the summary. n must be designated as a 2-digit decimal number. If this command is omitted, 8 spaces will be output. This command may be inserted anywhere in a summary.
- . LFnn..... Specifies the line spacing in the text. If not set, the line spacing will be single.
- . TITLE 'character string'..... This designates the title which is printed on each page of the text of the document.

[Example]

Please refer to Section 3.5 on the documentation generation functions.

---

```

; .                                TAG                                ; .

```

---

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
			; . [character string]

## [Function]

The character string which follows the ; . is registered as a TAG.

## [Application]

The character string registered may be used as a title for the lowest level of the programming hierarchy in SIMPLEHOST. It may also be output as a SUMMARY title in assembler lists.

## [Explanation]

- (1) The character string registered is the character string from the character immediately following the ; . to the next carriage return or line feed.
- (2) The maximum number of characters permissible in the character string is 255 ; if more than that number is entered, only the first 255 will be registered.
- (3) The TAG instruction may be entered in whichever position is desired.

## [Example]

Please refer to Section 3.5 on the documentation generation functions.

### 3.4 Macro Functions

Where the same routine is used a number of times in a program, it is possible to reduce the number of program steps by creating generalized subroutines. In cases in which routines are a little different and cannot be made into subroutines, or parameters are different, macro functions may be used to increase programming efficiency.

In this section, the use of macro functions is explained in detail, with examples. For methods of entering macro definition directives, please refer to Section 3.2.7.

#### 3.4.1 Macro definitions

The macro definition directives `MACRO` and `ENDM` are used to define macros. It is also possible to enter formal parameters when defining macros. The symbols which are defined within macros are of two types: local symbols which are effective only within the macro, and `GLOBAL` symbols which are effective in routines outside the macro.

To create a `GLOBAL` symbol, it is necessary to declare the symbol concerned globally within the macro using the `GLOBAL` directive. Symbols which are not globally declared are treated as local symbols effective only within the macro concerned. For the `GLOBAL` directive, please refer to Section 3.2.8 on symbol definition within macros.

## 3.4.2 Macro references

## (1) [Entry format]

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[Label:]	Name	<Actual parameter list>	[;comment]

## (2) [Function]

Refers to macro bodies defined by MACRO and ENDM statements.

## (3) [Explanation]

- i) The name is a MACRO name entered in the symbol column of a MACRO statement. It must be defined prior to the reference.
- ii) There are four types of formats which may be entered as actual parameters; these are evaluated as 16-bit data.
  - o Expressions
  - o Character constants  
(enclosed in quotation marks).
  - o Spaces (no entries, only commas).
- iii) The replacement of formal parameters by actual parameters is in accordance with the order of entry and in order from the left.  
 However, if the number of actual parameters exceeds the number of formal parameters, an O error (operand count error) will be generated.  
 If the number of actual parameters is less than the number of formal parameters, an empty character string will be assigned to the remaining formal parameters, and if a macro reference occurs, no error will be generated.  
 However, the empty character string will cause an error to be generated when the macro is developed.
- iv) If spaces, commas, quotation marks, semicolons or tabs and so forth are entered in actual parameters, they must be enclosed in quotation marks as character strings.
- v) It is possible to enter macro reference statements in macro bodies. Nesting, including the nesting of iteration

directives, built-in macro instructions, macro reference statements and IF statements, is possible up to a maximum of 40 levels. If this is exceeded, an N error (nesting overflow) will be generated, and the excess will not be assembled.

(4) [Example]

Referring to the previously defined macro (ADMAC):

```
      .  
      ADMAC 10H,20H  
      .
```

(Comments)

"ADMAC" is the macro name defined by a macro definition directive, while "10H, 20H" is the actual parameter required when making a reference to "ADMAC".

## UMAS17K ASSEMBLER

### 3.4.3 Macro expansion

Source programs which use macros are assembled in the following order.

- (1) If there is a macro definition, the macro body is stored as is in the internal assembler memory region (macro register).
- (2) Next, when a macro reference is discerned, the corresponding macro body is retrieved from the symbol table, and loaded into the macro name position.
- (3) Developed programs are assembled. However, if double semicolons have been entered in the macro body, everything after those double semicolons to the end of that line will be regarded as a comment within the macro definition, and will not be developed at the time of referencing the macro.

[Example]

ADMAC	MACRO	A1.A2.B1.B2	
	ADD	A2.B2	] ①
	ADDC	A1.B1	
	ENDM		
ADMAC		R0.R1.#0H.#1H	; ②

(Comments)

- (1) The macro with the name "ADMAC" is to be defined. A1, A2, B1, B2 are formal parameters.
- (2) The macro "ADMAC" is to be referred to. R0, R1, #0H, #01H are actual parameters corresponding to the formal parameters, A1, A2, B1 and B2. In this example, R0 and R1 must have been predefined as MEM type symbols.
- (3) The result of referring to ADMAC is developed in the manner shown below.

```
ADD R1.#1H
```

```
ADDC R0.#0H
```



### 3.4.4 Examples of the use of macros

#### Example 1:

An example of a macro definition.

PMAC	MACRO	P1,P2	; ①
L1	DAT	OCH	; ②
	DW	P1	
	DW	P2	
	.		
	ENDM		
	.		
L1	DAT	O4H	; ③
	.		
	.		
	PMAC	3000H,L1	; ④
	.		
	.		

#### (Comments)

- (1) P1 and P2 are formal parameters. A reference is being made to a DW operand column in a macro.
- (2) The symbol L1 is being defined in the macro. Since this symbol is handled as a local symbol, it will only be effective in this macro.
- (3) The symbol L1 is being defined in the main routine. The symbol L1 in the macro is a local symbol and a second definition cannot be made.
- (4) Here, there is an instruction referring to a macro called PMAC. 3000H and L1 are actual parameters corresponding to P1 and P2. L1 is treated as a symbol defined in a macro. If it is desired to send parameter L as a character constant, ''L'' is entered in the actual parameter entry operand.
- (5) The result of the macro reference is developed as shown below.

```
L1 DAT OCH
    DW 3000H
    DW L1
```



(Comments)

Since no actual parameter is entered in the macro reference statement, a NULL string is returned as the parameter, and an O error (operand count error) is generated in the IF statement.

Example 4:

First example of using a global symbol.

```
PERIOD MACRO P1,P2
        GLOBAL TIME1, TIME2          ; ①
TIME1 SET (10000/P1) AND OFFH
TIME2 SET (10000/P2) AND OFFH
ENDM
.
.
PERIOD 455,100                        ; ②
.
.
PERIOD 640,2400                       ; ③
.
.
```

(Comments)

- (1) TIME 1 and TIME 2 are defined as global symbols. In this example, P1 and P2 are formal parameters.
- (2) 455, 1000 is defined as a actual parameter.
- (3) 640, 2400 is defined as a actual parameter.
- (4) The macro reference is developed as set out below.

```
②    TIME1 SET 10000/455 AND OFFH
      TIME2 SET 10000/100 AND OFFH
③    TIME1 SET 10000/640 AND OFFH
      TIME2 SET 10000/2400 AND OFFH
```

Symbols defined using the SET directive may be defined twice or more. If symbols defined with DAT, MEM, FLG or LAB directives in macros are globally declared, and referenced twice or more in a macro, an S error (symbol multi defined) will be generated.

Example 5:

Second example of using a global symbol

```
STMAC MACRO
      GLOBAL SYMA
SYMA DAT OH
      DW SYMA
      ENDM
      STMAC
      .
      .
      DW SYMA
      MOV MEMOD,#SYMA
```

(Comments)

A symbol defined as a global symbol in a macro will continue to have that value even though macro development is complete.

### Example 6:

Third example of using a global symbol

```
BICMAC MACRO
    GLOBAL FLGA
    FLGA  FLG  0.10H.1
    FLGB  FLG  0.10H.2
    SET1  FLGA
    CLR1  FLGB
    ENDM
    BICMAC
    .
    .
    SKT1  FLGA
    SKF1  FLGB
```

### (Comments)

It is possible to refer to the symbol FLGA defined as a global symbol in a macro, but it is necessary to redefine the local symbol FLGB if it is to be used outside the macro.

### Example 7:

Using symbols defined outside macros.

```
DAT1 DAT 0
MEMO1 MEM 0.01H
SMMAC MACRO
    DAT1 DAT 1
        MOV MEMO1,#DAT1
    ENDM
    .
    MOV MEMO1,#DAT1
```

## UMAS17K ASSEMBLER

(Comment)

A symbol defined outside a macro may be used without change inside a macro. Further, as is the case with DAT1 in the example above, redefinition is possible within the macro. However, the value defined for the macro will only be effective within the macro, and the externally defined value will be the effective one outside the macro.

Example 8:

An example in which the limitations of global declarations are seen--even though the symbol was defined with a SET directive--with the result that the original value is the effective one outside the macro.

Assemble list:

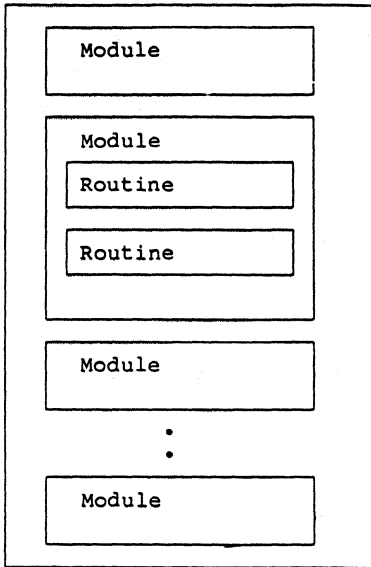
```
SYMB SET 0
SYMC SET 1
                                Macro development section
    PMAC
    GLOBAL SYMB
    SYMB SET 2
    SYMC SET 3
    MOV M,#SYMB
    MOV M,#SYMC
    .
    .
    MOV M1,#SYMB
    MOV M1,#SYMC
```

If SYMB and SYMC, which have been redefined within the macro, are referred to a second time outside the macro, the value 2 redefined for SYMB which has been globally declared is effective, but with SYMC, the original value 1 is the effective one.

### 3.5 Document Generation Functions

It is possible to generate lists with the AS17K's documentation generation control instructions. These lists are output separately from assemble lists, and are of three varieties: program summaries which summarize whole programs, module summaries which summarize each module, and routine summaries which deal with these modules in more detail and summarize routines. A table of contents is automatically generated for each of these types of lists.

#### Program



## UMAS17K ASSEMBLER

---

### 3.5.1 Program summaries

Program summaries may be obtained by designating the assemble option /SUMMARY. The SUMMARY option may be used to designate file names of the following formats to indicate where the titles and texts of these program summaries are stored.

[Entry format]

```
/SUM[MARY] = 'title', filename
```

All text in the file designated by the second argument of the option becomes the program summary text. The file designated by the SUMMARY option contains the text which summarizes the whole program.

[Example]

The SUMMARY option is designated as below.

```
/SUM= '0.0 ABSTRACT', PROG.SUM
```

In addition, the file PROG.SUM can be set to be contain the following data:

```
This program is ...
```

The use of this SUMMARY option permits text to be output in a list of the following sort:

```
0.0 ABSTRACT  
    This program is ...
```

[Points to be noted]

As is the case with the SUMMARY control instruction, ".EJECT" is used to create a page break. All commands which can be used with the SUMMARY control instruction can also be used here.



### 3.5.2 Module summaries

A module summary is designated by a SUMMARY control instruction which appears at the very beginning of each module. In the text of the module summary, the title and summary designated by the SUMMARY control instruction, plus a list of symbols declared publicly or externally in that module, plus the address range for that module, is output.

### 3.5.3 Routine summaries

Routine summaries are designated with the SUMMARY control instruction, and may be entered twice or more in each module. The "routine" referred to here indicates a statement which is entered between two or more SUMMARY control instructions or the next instruction with a title, or an END directive. Routine summaries contain the following information in addition to the title designated by the SUMMARY control instruction and the summary itself.

#### (1) ENTRANCES

A list will be output in alphabetical order of references from outside the routine to labels defined in the routine.

#### (2) MEMORIES CHANGED

MEM type symbols which carry out write operations in the routine are output alphabetical order.

The symbols referred to here are MEM symbols entered in the first operand of a transfer or operation instruction. If the symbol is operated on in the operand, the symbol is output with parentheses. Further, if a "MOV @R, XX" instruction is executed in a routine, the "@R" is treated as a symbol and output in this column.

#### (3) MEMORIES REFERRED

MEM type symbols which carry out read operations in the routine are output in alphabetical order.

The symbols referred to here are MEM symbols entered in the first operand of a transfer or operation instruction. If the symbol is operated on in the operand, the symbol is output with parentheses. Further, if a "MOV @R, XX" instruction is executed in a routine, the "@R" is treated as a symbol and output in this column.

**(4) MEMORIES MANIPULATED**

MEM type symbols entered using types and amendment functions in places where, in that routine, DAT types should be entered, are output in alphabetical order.

**(5) FLAGS CHANGED**

FLG type symbols entered in the operands of SETn/CLRn/NOTn built-in macro directives in the routine are output in alphabetical order.

**(6) FLAGS REFERRED**

FLG type symbols entered in operands of SKTn/SKFn built-in macro directives in the routine are output in alphabetical order. If operations occur in the operand, parentheses are attached when outputting.

**(7) DATA REFERRED**

All DAT type symbols which appear in a routine are output in alphabetical order. Parentheses are not given to the DAT type symbol if there is an operation.

**(8) TIBRANCH TO**

This outputs the contents of the operand column of a branch instruction used within the routine, and also the contents of comments in that line. However, this is limited to instructions which branch outside the routine. In the case of the direct branch instruction "BR @AR", @AR is regarded as a symbol and is processed in the same way. If an operation takes place in the operand, parentheses are attached when output.

**(9) SUBROUTINES CALLED**

This outputs, in alphabetical order, symbols with label attributes which are entered in a CALL instruction operand in that routine. If an operation has taken place, parentheses are attached when output. If there is a direct subroutine call instruction "CALL @AR", the @AR is treated as an operand and processed similarly.

**(10) LABEL MANIPULATED**

LAB type symbols which have been entered using the .DL. type conversion function in an operand in which a DAT type symbol should be entered are output in alphabetical order.

Parentheses are not output even though an operation may have taken place.

If this column list makes use of a direct branch instruction, a direct subroutine call or a look up table (MOVT @AR) instruction, an offset object results.

(11) SYSTEM CALL

This item is output only with assemblers for products which are provided with a SYSCAL instruction. This column outputs the SYSCAL instruction operand column entries as they are. (For example, even an operand containing numerals only will be output).

## UMAS17K ASSEMBLER

### 3.5.4 Examples

Program summaries are designated in the same way as the examples on the previous pages for the assemble option SUMMARY. On this occasion, the SUMMARY control instruction which appears at the very beginning of a module is entered in the following manner.

```

SUMMARY  '%','1.0 INITIALIZE MODULE'

        This module initializes all the variables ...

-----

%
```

A document list is output as follows:

```

0.0 ABSTRACT ]
              ] ①
        This program is .....

1.0 INITIALIZE MODULE ; ②'
                    ] ②
        This module initializes all ...
-----

PUBLIC(DAT):AA,AB ...,XYZ ] ③
PUBLIC(LAB):AYZ,BBC,...
EXTRN (LAB):EFG,...Z80 ... ; ④
ADDR.RANGE :0000H-0128H ; ⑤
```

(Comments)

- ① The SUMMARY option is used to designate the program title and summary.
- ② The SUMMARY instruction is used to designate the module title and summary.
- ③ The different symbol types publicly declared in the module are output alphabetically.
- ④ Symbol types which are the subject of EXTRN declarations in that module are output alphabetically.
- ⑤ The program memory address range occupied by that module is output as 4 hexadecimal digits.

If it is desired to omit section ②, the second operand in the SUMMARY control instruction is omitted. In this case, the table of contents page described below will not be generated. If it is not desired to output ②, a NULL string ( ' ' ) is designated as the first operand.

### 3.5.5 The table of contents generation function

Tables of contents may be automatically added to lists.

The table of contents gives the page numbers of lists for each title. This documentation generation function does not have the capability of stopping automatically a table of contents. Thus, when it is desired to end the table of contents, a space should be entered at the point in each SUMMARY control instruction statement where each title is designated.

[Example]

Outputting the example given in 3.5.4

TABLE OF CONTENTS		PAGE
0.0	ABSTRACT .....	XXX
1.0	INITIALIZE MODULE .....	XXX
1.1	TIMER INITIALIZE .....	XXX
1.2	PORT INITIALIZE .....	XXX

\* This line spacing is designated by the ".LF" command.

[Output example]

Program source list

SUMMARY'¥', ' 1.1 TIMER INITIALIZE'

THIS ROUTINE INITIALIZES MEMORIES ASSIGNED FOR CLOCK.  
 BEFORE THE INITIALIZATION, TIME SYSTEM IS CHECKED ;12-HOUR OR 24-HOURS.  
 IF 24-HOUR-SYSTEM, CLOCK IS INITIALIZED TO 0:00. AND IF 12-HOUR-SYSTEM,  
 INITIALIZED TO 12:00 AM.

¥

```

;
TIMER:                                ;ENTRANCE OF THIS ROUTINE
MOV  MINL,#0                          ;
MOV  MINH,#0                          ;
SKI1 F24HR                            ;IF 12-HOUR-SYSTEM,
BR   HR12                             ;THEN GOTO 12-HOUR PROCESS
MOV  HRL,#0                            ;
MOV  HRH,#BLANK                       ;BLANK CODE IS SET TO THE TEN'S DIGIT OF HOUR
BR   INITPORT                         ;END OF 24-HOUR-SYSTEM INITIALIZE.

HR12:
MOV  HRL,#2                            ;
MOV  HRH,#1                            ;
SET1 AMFLG                             ;SPECIFY AM

```

SUMMARY'¥'

NOTE:THIS ROUTINE DOES NOT INITIALIZE MEMORIES FOR INTERVAL TIMER.

¥

;END OF 12-HOUR-SYSTEM INITIALIZE.

SUMMARY'¥', ' 1.2 PORT INITIALIZE'

ALL THE PORTS ARE INITIALIZED TO LOW.

¥

;

INITPORT:

### Document list

#### 1.1 TIMER INITIALIZE

THIS ROUTINE INITIALIZES MEMORIES ASSIGNED FOR CLOCK.  
BEFORE THE INITIALIZATION, TIME SYSTEM IS CHECKED ;12-HOUR OR 24-HOURS.  
IF 24-HOUR-SYSTEM, CLOCK IS INITIALIZED TO 0:00. AND IF 12-HOUR-SYSTEM,  
INITIALIZED TO 12:00 AM.

NOTE: THIS ROUTINE DOES NOT INITIALIZE MEMORIES FOR INTERVAL TIMER.

ENTRANCES	:TIMER
MEMORIES CHANGED	:HRH, HRL, MINH, MINL
MEMORIES REFERRED	:-
MEMORIES MANIPULATED	:-
FLAGS CHANGED	:AM
FLAGS REFERRED	:F24HR
DATA REFERRED	:BLANK
BRANCH TO	:INITPORT ;END OF 24-HOUR-SYSTEM INITIALIZE.
SUBROUTINES CALLED	:-
LABELS MANIPULATED	:-

#### 1.2 PORT INITIALIZE

ALL THE PORTS ARE INITIALIZED TO LOW.





### CHAPTER 4 BUILT-IN MACRO DIRECTIVES

#### 4.1 An Overview of the Built-in Macro directives

In programs, macros which are defined in advance by the assembler are known as built-in macro instructions. The differences between a built-in macro instruction and a macro instruction defined by the user are set out below.

- (1) Built-in macro instruction development at assemble time is much faster than the time required for user-defined macro instructions.
- (2) If built-in macro instructions are developed when generating assemble lists, the statement at the point at which an object is generated is listed. However, if an SMAC, OMAC or NOMAC declaration has been made ahead of the built-in macro instruction, output is in accordance with that declaration.
- (3) Built-in macro instructions are optimized to create objects with the minimum number of steps.

There are 5 types of macro instructions as listed below.

SKTn, SKFn ...	Flag judgment
SETn, CLRn ...	Flag setting
NOTn ...	Flag inversion
INITFLG ...	Initialize flags
BANKn ...	Set banks

4.2 Built-in Macro Instructions

SKTn, SKFn  
 SETn, CLRn  
 NOTn  
 INITFLG  
 BANKn

The following points should be noted when making use of built-in macro instructions.

When an instruction which has a skip function is entered immediately before a built-in macro instruction, the instruction shown below is automatically entered in the development format so that processing does not lead to a program logic contradiction.

```
BR $+2
BR $+m+1 m: The number of steps in the statement developed
           by the built-in macro instruction.
```

However, if amending the source program, as is shown in the example below, it is possible to shorten the program by one step. Thus, if a BR instruction has been generated, a warning will appear in the assemble list (may be shortened BR), to indicate the potential for reducing the number of steps.

[Example]

In the following program, SET2, the instruction for setting the flags AFLG and BFLG is skipped, so that when the program is amended as shown below, there is no need for a BR instruction, and the length of the program can be reduced a step.

```
SKE M,#1      Development
SET2 AFLG,BFLG .....
.
.
.
```

```
SKE M,#1
BR $+2        ] Generated
BR $+5
PEEK WR,.MF.AFLG SHR 4
OR WR,#.DF.AFLG AND OFH
POKE .MF.AFLG SHR 4,WR
OR .MF.BFLG SHR 4,#.DF.BFLG AND OFH
```

```
SKNE M.#1  
BR NEXT      Development  
SET2 AFLG,BFLG .....  
NEXT:  
.  
.  
.
```

```
SKNE M.#1  
BR NEXT  
PEEK WR,.MF.AFLG SHR 4  
OR WR,#.DF.AFLG AND OFH  
POKE .MF.AFLG SHR 4,WR  
OR .MF.BFLG SHR 4,#.DF.BFLG AND OFH  
NEXT:
```

---

SKT <sub>n</sub>	SKIP IF <sub>n</sub> FLAGS ARE TRUE	SKT <sub>n</sub>
SKF <sub>n</sub>	SKIP IF <sub>n</sub> FLAGS ARE FALSE	SKF <sub>n</sub>

---

Symbol	Mnemonic	Operand	Comment
[label:]	SKT1	<symbol(FLG type)>	[;comment]
[label:]	SKT2	<symbol(FLG type)>, <symbol(FLG type)>	[;comment]
[label:]	SKT3	<symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>	[;comment]
[label:]	SKT4	<symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>	[;comment]
[label:]	SKF1	<symbol(FLG type)>,	
[label:]	SKF2	<symbol(FLG type)>, <symbol(FLG type)>	[;comment]
[label:]	SKF3	<symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>	[;comment]
[label:]	SKF4	<symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>	[;comment]

[Function]

SKT<sub>n</sub>: If all flags designated in the operand column are set to 1, skips the next instruction.

SKF<sub>n</sub>: If all flags designated in the operand column are set to 0, skips the next instruction.

[Application]

This is used to skip instructions, depending on the flag status.

[Explanation]

- (1) If a block other than <symbol(FLG type)> is entered in the operand column, an O error (illegal operand type) will be generated.
- (2) If a mixture of flags from different bank data memories are entered in the operand column, a B error (BANK unmatched) is generated.

- (3) Flags defined in data memory and in registers of files may both be entered in the operand column. The section in the register file where the data memory and addresses are stacked (40H~7FH) is processed as data memory.
- (4) A maximum of 40 levels of nesting is possible, including the nesting of iteration directives, IF statements, built-in macro instructions, and macro reference statements.
- (5) If the operand number and the value of n in SKTn and SKFn are different, an O error (operand count error) will be generated.

[Example]

A_flag	MEM	0.10H			
A1_flag	FLG	.FM.A_flag	SHL 4+1		
A2_flag	FLG	.FM.A_flag	SHL 4+2		
A3_flag	FLG	.FM.A_flag	SHL 4+4		①
D_flag	MEM	0.40H			
D1_flag	FLG	.FM.D_flag	SHL 4+1		
D2_flag	FLG	.FM.D_flag	SHL 4+2		
D3_flag	FLG	.FM.D_flag	SHL 4+4		
	SKT2	A1_flag, D3_flag			; ②
	BR	XY			
	SKF3	A2_flag, A1_flag, A3_flag			; ③
	BR	YZ			

(Comments)

The above is an example showing the use of SKTn and SKFn.

At ①, the flags to be used at ② and ③ are defined.

At ②, the flags\_A1 and\_D3 are set, "BR XY" is skipped and the next instruction is executed.

At ③, when the flags\_A2, \_A1 and\_A3 are cleared, "BR YZ" is skipped and the next instruction is executed.

	SET n CLR n	SET n FLAGS CLEAR n FLAGS	SET n CLR n
<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[label:]	SET1	<symbol(FLG type)>	[:comment]
[label:]	SET2	<symbol(FLG type)>, <symbol(FLG type)>	[:comment]
[label:]	SET3	<symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>	[:comment]
[label:]	SET4	<symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>	[:comment]
[label:]	CLR1	<symbol(FLG type)>	[:comment]
[label:]	CLR2	<symbol(FLG type)>, <symbol(FLG type)>	[:comment]
[label:]	CLR3	<symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>	[:comment]
[label:]	CLR4	<symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>, <symbol(FLG type)>	[:comment]

### [Function]

SETn: Sets all flags designated in the operand column to 1.

CLRn: Sets all flags designated in the operand column to 0.

### [Application]

Used to operate flags.

### [Explanation]

- (1) If a block other than <symbol(FLG type)> is entered in the operand column, an O error (illegal operand type) will be generated.
- (2) If a mixture of flags from different bank data memories are entered in the operand column, a B error (BANK unmatched) is generated.
- (3) Flags defined in data memory and in registers of files may both be entered in the operand column. The section in the register file where the data memory and addresses are stacked (40H~7FH) is processed as data memory.

- (4) A maximum of 40 levels of nesting is possible, including the nesting of iteration directives, IF statements, built-in macro instructions, and macro reference statements.
- (5) If the operand number and the value of n in SETn and CLRn are different, an O error (operand count error) will be generated.

[Example]

A_flag	MEM	0.10H	
A1_flag	FLG	A_flag.0	
A2_flag	FLG	A_flag.1	
A3_flag	FLG	A_flag.2	
	.		
D_flag	MEM	0.40H	
D1_flag	FLG	D_flag.0	
D2_flag	FLG	D_flag.1	
D3_flag	FLG	D_flag.2	
	.		
	SET2	A1_flag, D3_flag	; ②
	.		
	CLR3	A2_flag, A1_flag, A3_flag	③

(Comments)

The above is an example of the use of the built-in macro directives SET2 and CLR3.

At ①, the flags to be used at ② and ③ are defined.

At ②, flags\_A1 and \_D3 are set to 1.

At ③, flags\_A2, \_A1 and \_A3 are reset to 0.

NOT<sub>n</sub>

NOT

nFLAGS

NOT<sub>n</sub>

Symbol	Mnemonic	Operand	Comment
[label:]	NOT1	<symbol(FLG type)> ,	
[label:]	NOT2	<symbol(FLG type)> , <symbol(FLG type)>	[;comment]
[label:]	NOT3	<symbol(FLG type)> , <symbol(FLG type)> , <symbol(FLG type)>	[;comment]
[label:]	NOT4	<symbol(FLG type)> , <symbol(FLG type)> , <symbol(FLG type)> , <symbol(FLG type)>	[;comment]

## [Function]

Inverts all flags designated in the operand column.

## [Application]

Used to invert flags.

## [Explanation]

- (1) If a block other than <symbol(FLG type)> is entered in the operand column, an O error (illegal operand type) will be generated.
- (2) If a mixture of flags from different bank data memories are entered in the operand column, a B error (BANK unmatched) is generated.
- (3) Flags defined in data memory and in registers of files may both be entered in the operand column. The section in the register file where the data memory and addresses are stacked (40H~7FH) is processed as data memory.
- (4) A maximum of 40 levels of nesting is possible, including the nesting of iteration directives, IF statements, built-in macro instructions, and macro reference statements.
- (5) If the operand number and the value of n in NOT<sub>n</sub> are different, an O error (operand count error) will be generated.



[Example]

```
A_flag MEM 0.10H  
A1_flag FLG A_flag.0  
A2_flag FLG A_flag.1  
A3_flag FLG A_flag.2  
.  
D_flag MEM 0.40H  
D1_flag FLG D_flag.0  
D2_flag FLG D_flag.1  
D3_flag FLG D_flag.2  
.  
NOT1 A1_flag ; ②  
.  
NOT3 A3_flag, D2_flag, A1_flag : ③
```

(Comments)

The above example shows the use of the built-in macro directives NOT1 and NOT3.

At ①, the flags to be used at ② and ③ are defined.

At ②, the flag-A1 is inverted.

At ③, the flags-A3, D2 and A1 are inverted.

### INITFLG INITIALIZE 4 FLAGS INITFLG

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[label:]	INITFLG	----- ↓	[;comment]
		[NOT]<symbol(FLG type)>,[NOT]<symbol(FLG type)>, [NOT]<symbol(FLG type)>,[NOT]<symbol(FLG type)>	

#### [Function]

Initializes the four flags designated in the operand column to be set as 1 or reset as 0.

#### [Application]

Used to initialize the flags.

#### [Explanation]

- (1) It is necessary to enter four individual symbols (FLG type) in the operand column. If three or fewer or five or more FLG type symbols are entered, an 0 error (operand count error) will be generated. To set three or fewer flags, use the CLR and SET built-in macro instructions. To set five or more flags, use the INTFLG, CLR and SET built-in macro instructions together.
- (2) If "NOT" is entered before the symbol (FLG type) entered in the operand column, that flag will be reset to 0; if, "NOT" is not used, that flag will be set to 1.
- (3) If the memory addresses for the four flags are the same, they may be developed by a MOV instruction. In other cases, they will be developed by multiple step AND and OR instructions.
- (4) If a flag with a different bank is set in the operand column, a B error (BANK unmatched) will be generated.



BANK<sub>n</sub>BANK<sub>n</sub>BANK<sub>n</sub>

<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Comment</u>
[label:]	BANK <sub>n</sub>		[;comment]
		(0 ≤ n ≤ 15, n is a decimal integer)	

## [Function]

Sets the bank value in the bank register (address 79H).

## [Application]

This is used when it is desired to set or alter the bank register value.

## [Explanation]

- (1) For n, a decimal integer between 0 and 15 inclusive should be entered.
- (2) Banks may differ according to the product being used; if an attempt is made to designate a bank which cannot be used, a B error (invalid bank number) will be generated. Information on the memory size of each product is provided in the device file.
- (3) The BANK built-in macro instruction is developed as follows:  
BANK<sub>n</sub> → MOV BANK, #n ; the BANK is at address 79H.
- (4) If the bank built-in macro instruction is used in situations which permit the use of index registers, the bank register address may be modified depending on the contents of the index register. Please make sure that the index register is suppressed before using the BANK instruction.

[Example 1]

```
MOV IXH,#0000B      ] ①
MOV IXM,#0000B
MOV IXL,#0010B
SETI IXE            ; ②
BANK1               ; ③
```

(Comments)

The above is an example of the use of the BANK built-in macro instruction where bank switching is not possible.

At ①, the index register is set.

At ②, the index register is made available.

At ③, 1 may be set at address 7DH without setting the bank register (address 79H) 1 may be set at address 7DH.

[Example 2]

```
A1 MEM 1.10H      ] ①
A2 MEM 1.11H
.
.
BANK1             ; ②
MOV A1,#1         ] ③
MOV A2,#2
.
```

(Comments)

The above example illustrates the use of the BANK built-in macro instruction.

At ①, the memory address assigned to bank 1 is set. When using the memory assigned to bank 1, the bank register value must be set to 1, as at ②. From ③, the bank register value is 1.



**PART II OPERATIONS**





### CHAPTER 1 AN OUTLINE OF THE PRODUCT

#### 1.1 Details of the Product

Program name	Filename	Type of file
Assembler	AS17K.EXE	Command file

The command file is the first file read to memory when the program is started up.

#### 1.2 System Configuration

This section describes the operating environment required by the AS17K.

(1) Host machine

Refer to the Introduction for details of the personal computers which can be used.

(2) Operating system

PC-DOS (version 3.1)

(3) User memory size

512 K-bytes or more

(4) Files required for starting up the AS17K

① device file (UPD17XXX.DEV)

A file which stores instructions and mask options particular to individual products in the series which will be used in software development. Device files are purchased optionally for each product.

② source file (XXX.ASM)

The file which is assembled.

③ sequence file (XXX.SEQ)

A file which stores data for the designation of device filenames, assemble options, source filenames and so forth when the assembler starts up. Sequence files should be generated in advance if assembling multiple source module files.

④ MS-DOS environment setting file (CONFIG.SYS)

Setting values: Files = 15 (may be 15 to 20)

Buffers = 10 (may be 10 or more)



### CHAPTER 2 BEFORE EXECUTING

#### 2.1 Making Back-up Files

Before using the AS17K in actual assembly operations, the contents of all original disks should be copied to working disks in case the disks themselves or their contents are damaged. Original system disks should be carefully stored.

Order of generating back-up files:

- (1) Boot up MS-DOS
- (2) Insert MS-DOS system disk in drive A, and a new disk in drive B.
- (3) Use the FORMAT command to format the disk in drive B, and copy the system.

```
A>FORMAT B:/S ↓  
A>
```

- (4) Insert the AS17K system disk in drive A. Using the COPY command, transfer AS17K. EXE in drive A, and the device file D17000.DEV\* and with the sample program to drive B.

```
A>COPY A:*. * B:/V ↓  
A>
```

- (5) Next, transfer the drive A sample program to drive B. Before this is done, a subdirectory "¥SAMPLE" should be generated in a drive B file.

```
A>MD B:\SAMPLE ↓  
A>COPY A:\SAMPLE\*. * B:\SAMPLE/V ↓  
A>
```

(6) By this point, everything in drive A will have been transferred to drive B.

```
A>DIR B:/W↓  
  
    AS17K.EXE    D17000.DEV    <SAMPLE>  
  
A>DIR B:\SAMPLE ↓  
  
    MODULE1.ASM  
    MODULE2.ASM  
    MODULE3.ASM  
    SAMPLE.SEQ  
  
A>
```

Users should acquire separately a device file for the device which they are actually using. That device file (D17XXX.DEV) should be copied in the same way as is described above. D17XXX.OPT should also be copied at the same time for any device which requires the setting of mask options.

Sample programs are not currently supported.

### 2.2 Introduction to the Sample Program

Insert an MS-DOS system disk in drive A, and a disk containing a back-up file made up into drive B. Boot up MS-DOS.

```
A>DIR B:/W ↓
```

```
AS17K.EXE    D17000.DEV    <SAMPLE>
```

In this section, the following files are utilized as source files for the sample.

```
A>DIR B:\SAMPLE ↓
```

```
MODULE1.ASM  
MODULE2.ASM  
MODULE3.ASM  
SAMPLE.SEQ
```

AS17K.EXE is the assembler program itself.

UPD17000.DEV is the device file for the sample program.

Please do not use UPD17000.DEV for actual assembly operations.

**UMAS17K ASSEMBLER**

---

**2.3 Procedures for Assembling the Sample Program**

The sample program is not currently supported.

Two examples of input when booting up the assembler are given below:

(Example 1) Method of starting up the assembler itself

```
B>AS17K ↓  
UPD17000 SERIES ASSEMBLER V1.0  
.  
.
```

(Example 2) Designating sequence files when booting up the assembler

```
B>AS17K \SAMPLE\SAMPLE.SEQ ↓
```

For the details of sequence files, please refer to the entry formats in Part 2 Section 3.2 on sequence files.

**CHAPTER 3 SEQUENCE FILES****3.1 An Outline**

When booting up the assembler and carrying out assembly processes, it is first necessary to designate the device files, source module files and assemble options\* which are the subject of assembly processes. These are generically referred to as the assemble conditions. (There are two ways of designating assemble conditions: one by means of input from the monitor when booting up the assembler and another by designating the condition from a sequence file.) This section will deal with sequence files. If the designation from sequence file method is used, it is possible to designate multiple assembly conditions by invoking the one sequence file name. Further, during debugging, it is easy to delete or add in source module files simply by amending entries in the sequence file. As mentioned above, the use of sequence files can lead to more efficient debugging operations.

Note: Assemble options control whether or not lists are output when assembling. For details, please refer to Section 4.5 on assemble options.

### 3.2 Sequence File Entry Formats

Entries are made in sequence files by using an editor or the COPY command. The extension of a sequence filename must be ".SEQ".

#### 3.2.1 Total entry formats

```
[;comment]
    device file name                [;comment]; ①
    /option[/option/option/.../.../...] [;comment]; ②
    source file name [/option/.../.../...] [;comment]
    [
      .                               .
      .                               .
      .                               .
    ] [;comment] ③
    source file name [/option/.../.../...] [;comment]
```

#### [Explanation]

- (1) At ① the device filename is designated.
- (2) At ② the assemble option is designated. One item only of the assemble option may be designated between the two slashes. If designating multiple assemble options, enter them one after the other. If the designation of an assemble option occupies two lines or more, enter the slash at the beginning of the second and subsequent lines.  
The assemble option entered at ② is effective in the assembly of all source files.
- (3) At ③ the source module filenames plus the assemble options for each source file are designated. The entry format for the assemble option is the same as at ② above. If an assemble option for a source file entered at ② is of the same type as an assemble option in respect of all source files entered at ③ that option has priority.
- (4) When entering comments in a sequence file, semicolons should be used similarly to they are used in source programs.

Comments may be entered in any position in a file.





## 3.2.3 Assemble option entry formats

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

**[Function]**

Designates assemble options.

**[Explanation]**

- ① Assemble options are designated by entering them in the line after the line containing the device filename.
- ② Assemble option entries should start with a slash.
- ③ If multiple assemble options are designated, each option should be separated out with slashes. If a space or tab code is inserted between options, an error (invalid option) will be generated and assembling will halt.
- ④ Entries of two lines or more may be used for designating assemble options. If this is done, a carriage return or line feed should be inserted at the end of the line, and a slash entered at the beginning of the next line.
- ⑤ If conflicting assemble options are designated, the one entered later will be the effective one.
- ⑥ It is possible to omit the designation of an assemble option.
- ⑦ Please refer to section 4.5 on the assemble options for further details.

### 3.2.4 Source filename entry formats

```
source file name [/option [/option/.....] ] [;comment]
source file name [/option [/option/.....] ] [;comment]
:
source file name [/option [/option/.....] ] [;comment]
```

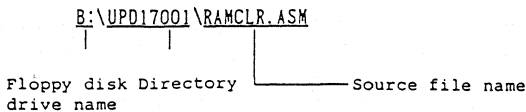
#### [Function]

Designates the names of the source files which will be assembled, and the assemble options which relate to them.

#### [Explanation]

- ① Two or more source filenames may be entered in one line. An assemble option corresponding to the source file may be entered after the source filename.
- ② When entering an assemble option after a source filename, be sure to include a slash preceding the assemble option. If a space is entered between the filename and the slash, an error will be generated.
- ③ If more than one assemble option is entered, they should be delimited by slashes.
- ④ A carriage return or line feed should be entered at the conclusion of designating assemble options in a source file.
- ⑤ The source file name entered can include an indication of the directory that contains that source file.

#### [Example]



- ⑥ Please refer to Section 4.5 on the assemble options for more details.

## UMAS17K ASSEMBLER

### 3.3 Sequence File Generation Methods

Verifying what is contained in SAMPLE.SEQ

```
A>TYPE B:\SAMPLE\SAMPLE.SEQ

D17000.DEV ; ①

/LIST=B:D17000.PRN/XREF=B:D17000.PRN
/HEX=B:D17000.HEX/ROW=50/MAP
/COLUMN=130/SUM/DOC ] ②

MODULE1.ASK/GEN
MODULE2.ASK/NOGEN ] ③
MODULE3.ASK/NOCOND
```

#### [Explanation]

At ① the device file which will be assembled is designated.

At ② the assemble options are designated.

③ indicates the source module to be assembled, and the assemble options for each source module. If an assemble option defined at ② is redefined at ③ the redefined option will be valid when assembling source files.

A sequence file is generated to assemble the sample programs named EX1.SEQ and EX2.SEQ. Sequence files may be generated in two ways: either by using an editor which operates under MS-DOS, or using the MS-DOS system command COPY. If the entry is short, the COPY command is adequate. However, the editor will be found to be more convenient for amending the type of sequence file described above or entering multiple designations. The procedure used in both cases is described below.

Sample programs are not currently supported.

- (1) Example of the COPY command.

```
A>COPY CON: B:EX1.SEQ ↓  
  
D17000.DEV ↓  
/LIST=B:D17000.PRN/XREF=B:D17000.PRN ↓  
/ROW=50/COLUMN=130 ↓  
MODULE1.ASM ↓  
MODULE2.ASM ↓  
MODULE3.ASM ↓  
~ 2  
  
A>
```

- (2) Example of using the editor.

The editor begins to operate at ① The filename is EX2.SEQ.  
At ② the details as shown below are entered using the screen editor. Care must be taken to input a carriage return or line feed.

```
D17000.DEV ↓  
/LIST=B:D17000.PRN/XREF=B:D17000.PRN ↓  
/ROW=50/COLUMN=130 ↓  
MODULE1.ASM/NOLIST ↓  
MODULE2.ASM ↓  
MODULE3.ASM ↓
```

At ③ the use of the editor is terminated.



### CHAPTER 4 ASSEMBLER FUNCTIONS

#### 4.1 Outline

The AS17K reads a designated source module file, and from the statements entered in that file, generates object files, assemble list files, memory maps and documents.

The AS17K assembler uses a two pass method. On the first pass, the symbol table is configured and the mnemonics are changed into machine language. An area is set aside for undefined symbols. On the second pass, machine language is assigned to the symbol region created during the first pass. On completion of the second pass, an interim object module file is generated. As it is an interim file, branch address data relating to more than one module files will not be determined.

The AS17K links interim object module files and generates an object file. The link process is executed automatically.

The AS17K is further provided with functions which shorten assembly time and make assembly operations more efficient. The date is registered in an interim object module file generated after completion of the second pass. When amending sections of source module files and reassembling, the dates of generation of the interim object module file and the source module file of the same name are compared, and the source module file will only be assembled if its date is later.

If the date of generation of the interim object module file is the later, the assembler concludes that the related source module file has not been amended, and that file will not be assembled.

In this situation, a link process is executed with a interim object module file which was already in existence when the assembler began operations.

**UMAS17K ASSEMBLER**

## 4.2 Input/Output Files

AS17K input files are as set out in the table below.

Input filename	Extension	File type
Device file	A file containing data peculiar to the device such as size of ROM and RAM, number of ports, or reserved words relating to register files.	. DEV*
Source file	A source file generated using an editor.	. <u>ASM</u>
Sequence file	A file in which are registered device files, assemble option designations and source module files. Assembling carried out by designating a sequence file is more efficient, obviating the need to designate device files and assemble options, or sort module files.	. SEQ
Mask option data file	A file which stores mask option designation data. The content varies depending on the type of product. It is not required with products which do not have option designations.	OPT*

\* Files supplied by NEC.

. \_\_\_\_ Note that source file extensions may be changed.



AS17K output files are as set out below:

Output file name type	Extension	File
HEX file	This is a file which contains Intel HEX format object codes, with IFL (internal function lists), and DFL (debug function lists). The Intel HEX format stop codes come after object codes and after IFL/DFLs.	<u>. HEX</u>
PROM file	This is a file which stores Intel HEX format object codes and IFL/DFLs. The Intel HEX format stop codes come after the IFL/DFLs. When downloading .PRO files from the PROM writer, write processing can be done by one write of the object code and IFL/DFL. (A PROM data file is used with an SE board.)	<u>. PRO</u>
Assemble list file	This file contains assemble lists for each source module file.	<u>. PRN</u>
Cross-reference list file	This file contains cross-reference lists for each source module file. However, if the list is not output, the extension will change to .XRF.	<u>. PRN</u>
Memory map file	This file automatically generates maps of data memory used and stores them as memory maps.	<u>. MAP</u>
Public cross-reference list file	This file stores cross-reference lists of symbols which have been publicly declared.	<u>. PUB</u>
Document file	This is a file which contains documents and module summaries generated by entering the documentation generation directive in source programs.	<u>. DOC</u>

**UMAS17K ASSEMBLER**

Output file name type	Extension	File
Report file	This file stores assemble reports	. REP
Logging file	This file stores error and warning messages output to the monitor during assembling. The name of this file is fixed: AS17K.LOG.	<u>. LOG</u>
Interim object module file	This is an interim file generated for each source module when assembled. It becomes an input file for the linking process.	<u>. OBJ</u>

\* \_\_\_ It should be noted that the extensions of all these files, .LOG and .OBJ excepted, can be changed.

### 4.3 Assembler Functions

#### 4.3.1 The interim object module file output function

When assembling starts, the designated source module file (.ASM file) is converted to machine language and an interim object module file with the same name as the source file is output as an .OBJ file. The time and date of output is registered in this interim object module file.

#### 4.3.2 The link function

The AS17K is an absolute assembler, but is provided with a link function in order to give it the capability of assembling source files split into modules. When a source module file is assembled, an interim object file is output corresponding to each source module file. Subsequently, when link processing takes place, this interim object module file becomes an input file.

#### 4.3.3 HEX file and PROM file output functions

The results of linking interim object module files are output as HEX and PROM files. The HEX file is divided into two sections: the object section and the IFL/DFL section, which are used when loading to the uPD17000 series development tool IE-17K. The PROM file is an SE board PROM data file. For further details, please refer to Part 2 Section 5 on assembler output lists.

#### 4.3.4 Functions which reduce assemble time

The AS17K is provided with functions which reduce assemble time, in the interests of more efficient debugging.

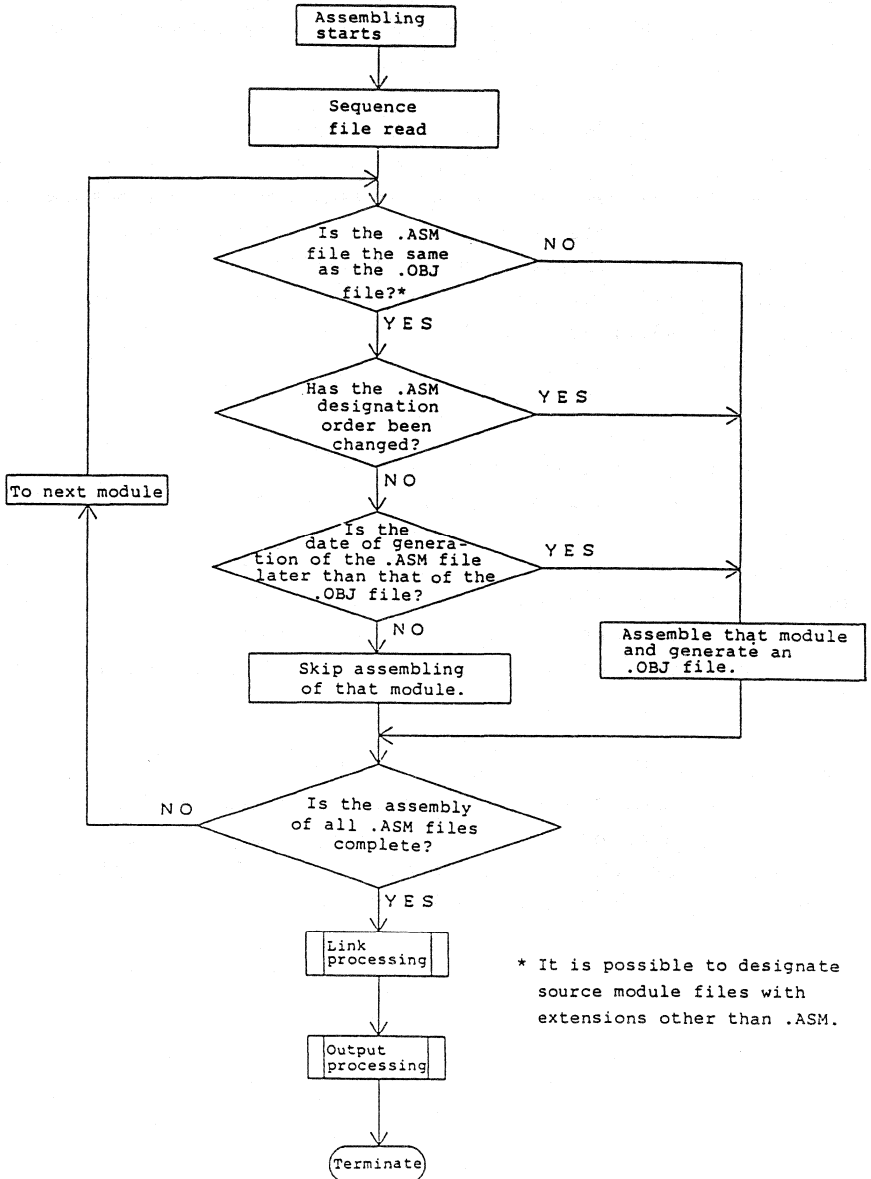
Prior to assembling source module files, the dates on which that source module file was generated is compared with the date of generation of the interim object module file of the same name. If the interim object module file generation date is later, the assembler concludes that the source module file of the same name has not been amended, and does not assemble it.

If the date of generation of the source module file is earlier than the corresponding date for the interim object module file of the same name, that source module file is assembled. Source module files designated after that source module file are all assembled without conditions.

Further, if the entry order of source module file designations is altered, added to, or deleted, after the initial assembling, files after the amended source module files are assembled without conditions.

The assemble time reduction function may be used effectively if debugged source module files are entered before source module files which are undergoing debugging.

Figure 4.1 Processing Procedures with Assemble Time Reduction Functions



#### 4.3.5 The assemble list file output function

An assemble list file is output once assembling is complete. Output control of assemble list files may be carried out through the assemble options. For more detail, please refer to Part 2, Section 5 on assemble output lists.

#### 4.3.6 The cross-reference list file output function

The AS17K generates two types of cross-reference list files. The first is a cross-reference list for each source module file, while the second is a cross-reference list for symbols which have been publicly declared (a public cross-reference list). Cross-reference lists which relate to the BR and CALL instructions may be included in assemble lists by using the assemble option BRANCH. For more details on this, please refer to Part 2, Section 5 on assemble output lists.

#### 4.3.7 The document file output function

The AS17K is provided with a function for outputting text documentation entered in source module files, using the documentation generation control instruction; this is output in the one file together with a table of contents. For more details, please refer to Part 1, Section 3.5 on the documentation generation functions, and Part 2, Section 5 on assembler output lists.

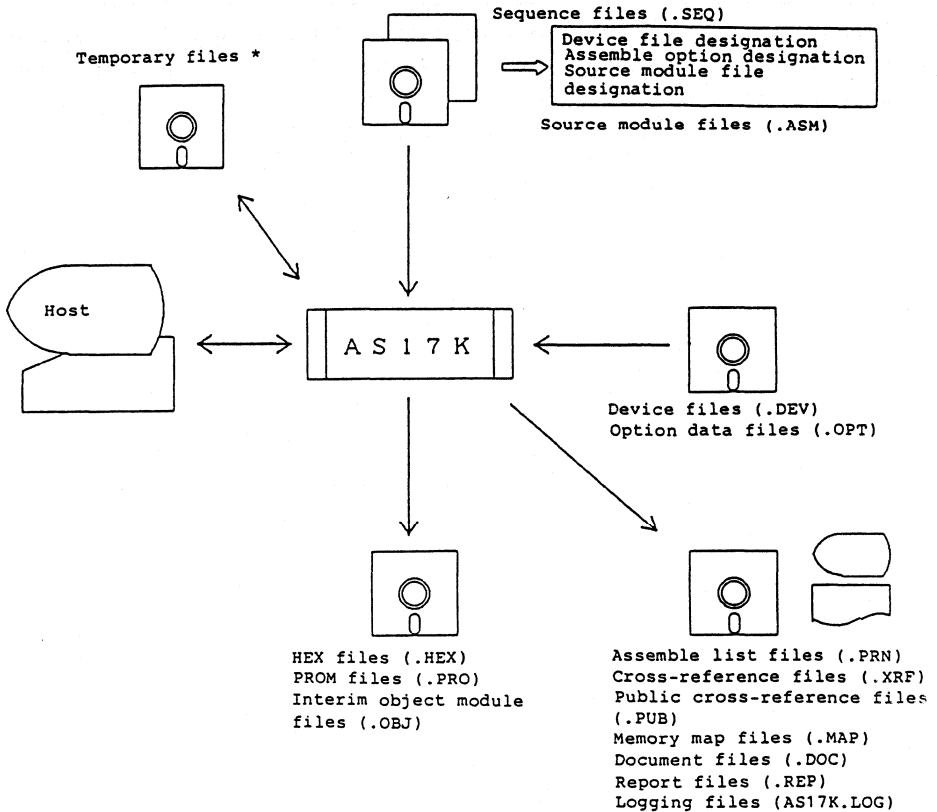
#### 4.3.8 The memory map file output function

The AS17K is provided with a function which automatically generates data memory maps, using source module files, and for outputting these as a single file. For more details, please refer to Part 2, Section 5 on assembler output lists.

### 4.3.9 The report file output function

The AS17K is provided with a function which outputs as a report file the situation with the use of host memory when assembling is being carried out, and the time required for assembling. For more details on this, please refer to Part 2, Section 5 on assembler output lists.

Figure 4.2 AS17K Input/output files



\* Temporary files are deleted when assembling is complete. They are distinguished by having extensions of the form .\$\$\$.

```
(AS1X. $$$, AS1P. $$$
AS1B. $$$, AS1M. $$$
AS1D. $$$, AS1S. $$$
AS1L. $$$)
```



### 4.4 Methods of Starting Up the Assembler

#### 4.4.1 Files which must be input when starting up the assembler

The following input files are needed in order to start up the assembler.

(1) Source module file (.ASM)

This is the file in which the source program is entered.

(2) Device file (.DEV)

This is a file prepared for each member of the series defining the data on ROM/RAM volume and so forth. In addition, this file defines flag names and so forth in register files.

(3) Mask option data file (.OPT)

This is a file which defines data which relates to mask options. If using device files for products which have mask options, this file will, when copied to the current drive, be automatically read during assembling operations. It is not needed with members of the series which do not use mask options.

For further details please refer to Part 2, Section 4.2 on assembler input/output files.

#### 4.4.2 Convenient input files

Sequence files (.SEQ)

This is a file in which is entered assemble options, device filenames and source program filenames required for assembling. The use of sequence files in assembling is recommended in order to improve assembling efficiency.

## UMAS17K ASSEMBLER

---

### 4.4.3 Methods of starting up the assembler

The actual procedures used to start up the assembler are explained below.

There are three different methods of inputting commands to start up the assembler.

#### Input methods

1.

2.

3.

Each of these three methods is explained separately in this section.

In order to start up the assembler most efficiently, the use of methods 1 and 2 set out above is recommended, together with the use of a sequence file.

1.  Start up

The disk containing the assembler and the device file is inserted in drive A, and the disk with the source files is inserted in drive B. "AS17K" is entered at the prompt (A>).

The assembler initiates a memory load operation.

After assembler operation is initiated, the current directory is searched for sequence files, and the operations described below are effected.

- ① If there is a single sequence file in the current directory That sequence file is automatically read, and assembling is carried out on the basis of what it contains.
- ② All sequence file names are allotted a number in order from one, and a list is displayed on the monitor. The numbers of sequence files to be assembled can be input here. If no sequence files are selected, inputting a carriage return or a line feed will return the user to the mode described below in ③ .

[Example]

```
A>AS17K ↓
UPD17000 SERIES ASSEMBLER
      COPYRIGHT (C) NEC CORPORATION. 1987,1988

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

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

SEQ FILE ? (SELECT NUMBER) = 2 ↓

TESTPRO1.ASM << ASSEMBLE START >> 11:24:30 04/11/88
```

- ③ If there are no sequence files in the current directory If there are no sequence files or if there are no files number selected in ② above, a carriage return or line feed may be input; when the prompt is displayed on the monitor as below, the appropriate filename can be input.

(i) Designating a device file.

- (A) If there is a device file in the current directory When a list of device files is displayed and a message "DEV file number:" is displayed, a number may be selected and input.

DEV file number:

If there is no appropriate device file, it is possible by inputting a carriage return or a line feed to move to a prompt which allows a device filename to be designated.

## UMAS17K ASSEMBLER

- (B) If there is no device file in the current directory  
A device file name may be input after the display  
shown below: The extension .DEV must be input even in  
situations in which files can normally be designated  
with the extension omitted.

DEV file name [.DEV]:

If a file name with an extension other than .DEV is  
designated, the message

"invalid file extension name"

will be output, and a display will appear prompting  
the user to enter another device filename.

Example 1 When there are two device file names in the  
current directory.

```
A>AS17K ↓
UPD17000 SERIES ASSEMBLER
  COPYRIGHT (C) NEC CORPORATION. 1987,1988

=== DEV FILE LIST IN CURRENT DIRECTORY ===

1) UPD17102.DEV    2) UPD17051.DEV

DEV file number : 2 ↓

Assemble options : /LIS/NOT/HEX ↓
Source file name : B:TESTPRO2.ASM ↓

TESTPRO2.ASM << ASSEMBLE START >> 11:24:30 04/11/88
```

Example 2 When there is no device file in the current directory.

```
A>AS17K
UPD17000 SERIES ASSEMBLER E2.0D (17 Feb 88)
  COPYRIGHT (C) NEC CORPORATION. 1987,1988

DEV file name : B:\D17000\D17001.DEV ↓
Assemble options : /LIS/NOT/HEX ↓
Source file name : B:TESTPRO2.ASM ↓
TESTPRO2.ASM << ASSEMBLE START >> 11:24:30 04/11/88
```

(ii) Designating assemble options

The next message output after completion of device file designation is:

Assemble options :

Assemble options may be input here. The input format is:

`/<option>[/<option>.....]`

Assemble options should be designated using the same entry format as is used in sequence files. However, it is not possible in this situation to enter an option in two lines or more. For details, please refer to Section 3.2.3 on the assemble option entry formats. If no option is to be designated, a carriage return or line feed should be entered.

(iii) Designating source filenames

The next message to be output after completion of assemble option designation is:

Source file name:

At this point source filenames and directories may be input. The extension may be omitted.

If the extension is omitted, the extension ".ASM" will be used automatically. It is possible to designate options following the source file name. After a source file has been designated, and the option relating to it entered, a carriage return or line feed is input. At this point assembling will start.

Note that it is not possible to designate multiple sourcefiles.

Should any error be made in designations in (i), (ii) or (iii) above, an error message will be displayed once assembling begins, and assembling will halt. If it is desired to halt input and terminate assembling, a control C should be input.

2.

The disk containing the assembler and the device file should be inserted in drive A, with the disk containing the sequence files going into drive B.

At the prompt (A>), input

After this has been done, the assembler will be loaded to memory, and assembling will begin in accordance with a sequence file named SAMPLE.SEQ which is contained on the disk in drive B.

It is possible to enter sequence files without the extension ".SEQ." This extension will be automatically assigned to a sequence file.

Sequence files are files which contain pre-recorded device file names, assemble options and source filenames.

For methods of generating sequence files, please refer to Section 3 in Part 2.

[Input example]

```
A> AS17K B:SAMPLE.SEQ ↓  
Output list
```

3. A>AS17K Δ <device file name> Δ [ / <option> [ / <option> / ..... ] ] Δ <source file name> [ / <option> / ..... ]

The disk containing the device file and the assembler is inserted in drive A, and the disk containing the source files is inserted in drive B. At the prompt (A>), AS17K is input, and on the same line are entered device filenames, assembler options and source filenames. The assembler will start assembling.

[Input example]

```
A>AS17K B:D17000.DEV /NOL/NOX B:MODULE1.ASM/LIS=MODULE1.PRN ↓  
Output list
```

After this is input, the assembler will be loaded into memory and assembling will begin in accordance with the device files, assemble options and source files. The directory in which a device file is contained may also be entered at the point at which a device file is asked for. The extension .DEV may be omitted. If it is omitted, an extension will be automatically assigned to the file. If no directory is specified, the current directory will be used.

Assemble options are entered in the option blocks in the command line given above. As with device filenames, assemble option specifications should be delimited by spaces. An assemble option entry must begin with a slash. If multiple assemble options are designated, each should be delimited with two slashes. Spaces and tabs should not be included in assemble option specification statements. If spaces or tabs are inserted, the assembler will interpret these as the termination of an assemble option entry.

The command line given above provides for the entry of the names and directories of source files to be entered, plus the assemble options related to those source files. The assemble option entry format is the same as given in the above examples of assemble options. It is to be noted that there are limitations on the varieties of assemble options which may be used with particular source files. Only one source file may be designated. For more details, please refer to Part 2, Section 4.5 on assemble options.

[ Example ]

```
A>AS17K UPD17001.DEV /PRM/ROW=70/WOR=F: TESTPRO5.ASM ↓
UPD17000 SERIES ASSEMBLER
      COPYRIGHT (C) NEC CORPORATION. 1987

TESTPRO5.ASM << ASSEMBLE START >> 11:28:00 12/24 87
```



### 4.4.4 Halting during assembly

If it is desired to halt assembly operations after they have begun, a control C should be input from the keyboard. When the assembler receives a control C, it will close all open files, terminate assembling, and return to the MS-DOS system prompt.

#### 4.5 Assemble Options

Assemble options are used to designate working drives, functions, and file formats for files to be output when assembling.

There are two methods of designating assemble options: they may be entered in sequence files, or input from the keyboard when assembling is started. For more details, please refer to Part 2, Section 4.4 on methods of starting up the assembler.

If no assemble options are designated at all, the preset default values for the assemble options will be used. It is also possible to distinguish between designations of assemble options which are effective for all assembler operations, and those which are effective for specific split module source files. However, only a limited number of the assemble options is validly usable in designations for all source files. For assemble options designable for all source files, please refer to the list of designations for each module in the assemble option list.

Table 4.1 List of Assemble Options

Option name	Default	Description	Designable or all modules *
NOO[BJECT] HEX,PROM	HEX	Object(load module file)	
NOL[IST] LIS[T]	LIS	Assemble list output control	0
NOX[REF] XRE[F]	XRE	Cross-reference list output control	0
ERR[OR] NOE[RROR]	NOE	Error skip control	0
ROW	ROW=66	List output page line number control	
GEN NOG[EN]	NOGEN	Control of development in macro and iteration instruction lists	0
COL[UMN]	COL=80	List output column number control	
AUT[OLOAD] NOA[UTOLOAD]	NOA	Automatic load control	
CON[D] NOC[OND]	NOCON	Print control for conditional statement lists	0
NOS[EQ] SEQ	SEQ	Option data output control	0
MAP NOM[AP]	NOM	Data memory map output control	
DOC[UMENT] NOD[OCUMENT]	NOD	Document output from assemble list control	
BRA[NCH] NOB[RANCH]	NOB	Buried cross-reference output control	
PROG	-	Program comment control	
TAB NOT[AB]	NOT	Tab control	
FOR[M] NOF[ORM]	NOF	Form feed control	

REP[ORT] NOR[EPORT]	NOR	Report output control	
ZZZn	ZZZn=0	Assemble variables (numeric characters from 0 to 9)	
PUB[XREF] NOP[UBXREF]	NOP	Public cross-reference list output control	
WOR[K]	Current drive	Operating drive designation	
SUM[MARY]	-	Program summary output control	
HOS[T] NOH[OST]	NOH	SIMPLEHOST data output control	

\* Options which can be designated for all modules are also options which can be designated for all source files.

### 4.5.1 The object file output control option

[Entry format]

[ { NOO[BJECT] HEX[=File name ] } ]
--

[Function]

Controls whether or not an object file is output; if object files are to be output, designates the file names to be output. Both HEX and PROM files may be designated independently.

[Explanation]

- (1) NOO[BJECT]: Option used when object files are not to be output.
- (2) HEX: Option for outputting HEX files to be downloaded to the IE-17K.
- (3) PROM: Option for outputting PROM files to be used when generating PROMs on the SE board.

**(4) File designation**

Output filenames may be designated after HEX or PROM.

Filenames are entered using the format "[drive name:\directory name\] filename"

If no output file name is entered, the directory and filename output will be as shown below.

- ① When using sequence files  
The sequence file and the same directory  
Files with the same name as the sequence file and the extensions .HEX or .PRO.
  
- ② If there is one source file to be assembled (a sequence file is not used)  
The source file and the same directory  
Files with the same name as the source file and the extensions .HEX or .PRO.  
It is possible to enter output filenames without entering the extension when designating them. In this situation, the output file extension .HEX or .PRO will be automatically assigned. It is also possible to enter in filename entry columns the MS-DOS reserved names AUX, CON, PRN and NUL. The order of output is as given below:

AUX ... RS-232C  
CON ... Console  
PRN ... Printer  
NUL ... No file output

[Default value]

HEX

### 4.5.2 The assemble list file output control option

[Entry format]

<pre>[ { LIS[T][=Filename [,PRN]] } ]   NOL[IST]</pre>
--

[Function]

Controls whether or not assemble list files are output, and designates the file name of the files to be output.

[Explanation]

(1) LIS[T]

The option used to output assemble list files.. There are two methods, as set out below, of designating what is to be output.

Without entering filenames

Generates a source filename and file with that name in the same directory as the source file. If the source program is split into modules, generates a file with the same name in the directory containing the source module file. The extension is .PRN.

File name entry

Generates a file with the designated filename. It is possible to use AUX, CON, PRN or NUL in the filename. The filename entry format "[drive name:[directory]filename" should be used. If an extension is not entered, the extension .PRN will be assigned.

(2) NOL[IST]

The option used if an assemble list file is not to be output.

**(3) [, PRN]**

If ", PRN" is entered following the entry of a filename, that file will be output to the printer at the same time as it is output. However, if LIST=PRN or PRN are entered, an error will be generated.

- (4) If a filename is designated in an option line when carrying out split assembling, a number of lists may be output in the one file.

[Default value]

LIST



### 4.5.3 The cross-reference list file output control option

[Entry format]

```
[ { XRE[F][=Filename [.PRN]] } ]  
  NOX[REF]
```

[Function]

Controls whether or not a cross-reference list file is output, and if a file is to be output, designates the name of that file.

A cross-reference list displays all line numbers which reference symbols and the line number, type, and value of each symbol in a source program. One cross-reference file is output for one source module file.

[Explanation]

(1) XRE[F]

The option used to output a cross-reference list file. There are two methods of designating files to be output, as given below.

Entry without filename

- ① When an assemble list is output, it is output together with the file which outputs the assemble list. Accordingly, the filename is the same as that of the assemble list.
- ② If the assemble list is not output, that is to say when NOL is designated, a file is output with the same name as the source file in the directory of the same name as the source file. In this case, the extension used is .XRF.

**Filename entry**

Generates a file with the file name designated. This is used when it is desired to output a file which is different from the assemble list, or if it is desired, when split assembling, to have one file with cross-reference lists output to multiple files. It is possible to enter as file names AUX, CON, PRN and NUL. Filename entry should take the format "[drivename:[directory]] filename. If entry of the extension is omitted, the extension .XRF will be assigned.

**(2) NOX[REF]**

This option is used when a cross-reference list file is not to be output.

**(3) [, PRN]**

If ", PRN" is entered following entry of a filename, the file will be output to the printer also. However, an error will be generated if XRE=PRN or PRN are entered.

**(4) Other cross-reference lists are buried format cross-references to public cross-references and assemble lists. For more details on designating these, please refer to Section 4.5.14 on the buried cross-reference output control option.**

[Default value]

XREF

### 4.5.4 The error skip control option

[Entry format]

[ { ERR[OR] } ] [ { NOE[RROR] } ]
--------------------------------------

[Function]

This option controls whether assembling is to continue or halt when an error message is generated during split assembling.

[Explanation]

(1) ERR[OR]

If an error is generated during an assemble operation, once that source module file has been assembled, assemble and link processing of the next source module file will be stopped.

(2) NOE[RROR]

Assembling will continue even though an assemble error is generated. The object relating to the statement which generated the error will be "074F0H" (NOP instruction).

[Default value]

NOE[RROR]

4.5.5 The list output page row number control option

[Entry format]

[ ROW=n ]

[Function]

This option designates the number of lines in a page for all list files (assemble lists, memory maps, cross-references, and so forth).

[Explanation]

n indicates the number of lines in one page. The number is entered in decimal, and must be within the range  $50 \leq n \leq 250$ .

[Default value]

ROW=66.

### 4.5.6 The macro iteration directive development control option

[Entry format]

```
[ { GEN      } ]  
  { NOG[EN] } ]
```

[Function]

This option controls whether or not statements which contain entries made by macro and iteration directives are to be developed in assemble lists and output, and whether output is to be in accordance with the macro development print control instructions of the assemble control instruction entered in the source.

[Explanation]

(1) GEN

All statements entered with macro and iteration directives are developed, and output to an assemble list. The macro development print control instructions have no effect.

(2) NOG[EN]

The development of statements entered with macro and iteration directives is in accordance with the macro development print control instructions LMAC, SMAC, OMAC and NOMAC.

[Default value]

GEN

**UMAS17K ASSEMBLER**

---

**4.5.7 The list output column number control option**

[Entry format]

[ COLUMN]=n ]
---------------

[Function]

This option specifies the number of columns in the line for all types of lists.

[Explanation]

The number of columns is expressed in decimal integers, and must be within the range 72 to 132.

[Default value]

COLUMN=80.

### 4.5.8 The automatic load control option

[Entry format]

```
[ { AUT[OLOAD] } ]  
[ { NOA[UTOLOAD] } ]
```

[Function]

This option controls whether or not a HEX file is to be sent automatically to the IE-17K during assembling.

[Explanation]

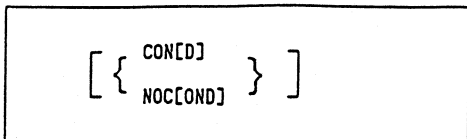
- (1) AUT[OLOAD]  
Sends a HEX file automatically to the IE-17K while assembling.
- (2) NOA[UTOLOAD]  
Does not send a HEX file to the IE-17K when assembling.
- (3) The automatic load control option is only effective when HEX file output has been designated with the object output control option.
- (4) If the IE-17K is not connected to the host when AUTOLOAD is selected, the HEX file is not automatically sent.

[Default value]

NOAUTOLOAD

## 4.5.9 The conditional statement output control option

[Entry format]



[Function]

This option determines whether statements entered with the assemble directive with conditions are to be output to the assemble list unconditionally, or in accordance with the output list control instructions.

[Explanation]

(1) CON[D]

Outputs statements entered with assemble directives with conditions to the assemble list without reference to the output list control instructions SFCOND and LFCOND.

(2) NOC[OND]

Outputs to the assemble list statements entered with the assemble directive with conditions in accordance with the output list control instructions SFCOND and LFCOND.

[Default value]

COND



### 4.5.10 The optional data output control option

[Entry format]

```
[ { NOS[EQ] } ]  
  SEQ
```

[Function]

This option controls whether or not the following details are output to the first page of the assemble list for each source module.

- ① Details entered with the program comment output control option (PROG=)
- ② Character strings entered on the same line following the command line for starting the assembler (command line option= ).
- ③ Sequence filenames and the details entered in sequence files designated when beginning assembly operations (SEQ= .).
- ④ The assemble option list designated when starting assembly operations.

[Explanation]

- (1) NOSEQ  
No option data is output to page 1 of the assemble list file.
- (2) SEQ  
Option data is output to page 1 of the assemble list file.
- (3) If NOLIST has been specified with the assemble list file output control option, this option does not apply.

[Default value]

SEQ

## 4.5.11 The data memory map file output control option

[Entry format]

<pre>[ { MAP[=Filename [,PRN]] } ]   NOM[AP]</pre>
--

[Function]

This option controls whether or not MEM and FLG type symbols defined in a source program, plus their corresponding data memory addresses (flag positions), are output as a map.

[Explanation]

## (1) MAP

A data memory map file is output.

## (2) NOM[AP]

Does not output a data memory map file.

## (3) Filename designation

An output filename may be designated following MAP.

The filename should be entered in the format

"[drive:[\directory\]filename". It is possible to omit entry of an output filename. In this situation the filenames and directories output will be as follows:

## ① When using sequence files

- sequence file and the same directory
- sequence file and files of the same name with the extension .MAP.

- ② When there is one source file to be assembled (a sequence file is not used) .
- the source file and the directory of the same name
  - the source file and files of the same name with the extension .MAP.

The extension may be omitted when entering designations of output filenames. In this situation, the output file will be automatically assigned the extension .MAP. It is also possible to use AUX, CON, PRN and NUL in entering file names.

(4) [, PRN]

If ", PRN" is input, the map list file is also output to the printer. However, an error will be generated if MAP=PRN or PRN are entered.

[Default value]

NOMAP

4.5.12 The document file output control option

[Entry format]

```

[ { NOD[OCUMENT]
  DOC[UMENT][= Filename [,PPN]] } ]
    
```

[Function]

This option controls whether or not a document is output.

[Explanation]

(1) NOD[OCUMENT]

Does not output a document file.

(2) DOC[UMENT]

Outputs a document file.

(3) Filename designation

An output file name may be designated after DOC. The filename entry format should be "[drive name:[directory name\] filename". AUX, CON, PRN and NUL may also be entered in filenames. If no filename has been entered to be output, the directory and filename which will be output is as given below.

① If a sequence file is used

The sequence file and the same directory.

The sequence file name and the name of the same name with the extension .DOC.

- ② If there is one source file to be assembled (a sequence file is not used)  
The source file and the same directory  
The source file name and files of the same name with the extension .DOC.  
It is possible to omit entry of the extension when designating the output filename. In this situation, the output file will be automatically assigned the extension .DOC. It is also possible to use AUX, CON, PRN and NUL in entering filenames.

(4) [, PRN]

If [, PRN] is entered after the filename designation, the document file will be output to the printer also. However, if DOC=PRN or PRN is entered, an error will be generated.

[Default value]

NOD[OCUMENT]

## 4.5.13 The report file output control option

[Entry format]

[ { REP[ORT][=Filename [.PRN]] } ] NOR[EPOR]
---

[Function]

This option controls whether or not to output as a report file data on the amount of memory used and time taken by the assembler to generate an output file.

[Explanation]

(1) REP[ORT]

Outputs a report file.

(2) NOR[EPOR]

Does not output a report file.

(3) Designating file names

A filename may be designated following REP[ORT]. The filename entry format should be "[drive name: [/directory name/]filename". It is also possible to enter AUX, CON, PRN and NUL in filenames. If no output filename is entered, the directory and filenames output shall be as follows:

- ① If a sequence file is used  
The sequence file and the same directory.  
The sequence file and the files of the same name with the extension .REP.
- ② If there is one source files to be assembled (a sequence file is not used)  
The source file and the same directory  
The source file name and file of the same name with the extension .REP.  
It is possible to omit entry of the extension when designating the output filename. In this situation, the output file will be automatically assigned the extension .REP. It is also possible to use AUX, CON, PRN and NUL in entering filenames.

(4) [, PRN]

If [, PRN] is entered after the filename designation, the document file will be output to the printer also. However, if REP=PRN or PRN is entered, an error will be generated.

[Default value]

REP[ORT]

## UMAS17K ASSEMBLER

### 4.5.14 The buried cross-reference output control option.

[Entry format]

[ { BRANCH } ] [ { NOBRANCH } ]
------------------------------------

[Function]

This option controls whether or not, when an address list is generated, address data contained in the BR and CALL instructions in the line prior to one containing the label referenced by the BR and CALL options is to be output.

[Explanation]

- (1) BRANCH  
Outputs a buried cross-reference to the assemble list.
- (2) NOBRANCH  
Does not output the buried cross-reference.
- (3) Output example (assemble list)

<u>Address</u>	<u>Symbol</u>	<u>Mnemonic</u>	<u>Operand</u>
:		:	
0010		BR	TABLE1
:		:	
0020		CALL	TABLE1
:		:	
:		:	
:		:	
:		:	
0100	TABLE1:	NOP	
:		:	

:B-0010,C-0020 ←



- (4) A cross-reference output when a BRA[NCH] designation is made is of the form B-XXXX where the reference follows a BR instruction, and of the form C-XXXX when the reference follows a CALL instruction.
- (5) This option is not effective if the assemble list file output control option NOL[IST] has been set.

[Default value]

NOB[RANCH]

**UMAS17K ASSEMBLER**

---

**4.5.15 The program name output control option**

[Entry format]

[            PROG="Program name "            ]
--

[Function]

This option designates a character string to be output to the program name column of an assemble option list, assemble list or cross-reference list. The character string should be enclosed in double quotation marks.

[Explanation]

(1) Entry format

The character string may contain a maximum of one to seven characters enclosed by double quotation marks. If eight characters or more are entered, an error is generated and assembling will halt.

Program name output control option entries are only possible in sequence files. If the attempt is made to use this option in other situations, the invalid option error will be generated.

(2) If the program name output control option is omitted, the following will be output to the program name column.

- ① If using a sequence file: the sequence file name.
- ② If not using a sequence file: the directory in which the source file is.

### 4.5.16 The TAB control option

[Entry format]

[ { TAB } ]
[ { NOT[AB] } ]

[Function]

This option controls whether, when assemble lists are generated, a tab code is to be output, and whether a space is to be inserted at the tab code position as if the next character after the tab code were eight column spaces from the beginning of the line.

[Explanation]

(1) TAB

Outputs a TAB code.

(2) NOT[AB]

See above.

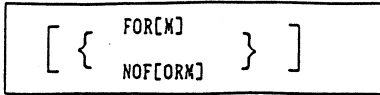
(3) This option should be used with printers which do not recognize tab codes. If tabs are selected, assembling speed will be quicker, and the volume of the files used will be smaller.

[Default value]

NOT[AB]

## 4.5.17 The form feed control option

[Entry format]



[Function]

This option controls whether to introduce a page break in an output list by a form feed code (the 8-bit JIS code 0CH) or the carriage return/line feed code.

[Explanation]

## (1) FOR[M]

The output list page break is inserted by a form feed code.

## (2) NOF[ORM]

The output list page break is inserted by outputting carriage returns/line feeds as far as the line designated by the list output page line number control option (ROW).

## (3) This option should be used with printers which are not able to recognize form feed codes. If FOR[M] is selected, assemble time can be speeded up and the volume of the files used can be reduced.

[Default value]

NOF[ORM]

### 4.5.18 The assemble variable control option

[Entry format]

[ ZZZn=m ]	n: An integer from 0 to 9. m: A value between 0H and 0FFFFH.
------------	---

[Function]

This option sets the value (m) of the assemble variable ZZZn when assembling starts.

[Explanation]

- (1) There are ten types of assemble variable: ZZZ0 ZZZ9; each may be set independently.
- (2) Each assemble variable may be set to 0H 0FFFFH. Do not use H if the display is decimal. In the case of a binary display, B should be used in place of H.

[Default value]

ZZZ0 ZZZ9 all 0  
(ZZZn=0)

### 4.5.19 The public cross-reference list file output control option

[Entry format]

```
[ { NOP[UBXREF] } ]  
[ { PUB[XREF][= Filename [.PRN]] } ]
```

[Function]

This controls whether or not a publicly declared symbol cross-reference list file is output.

[Explanation]

(1) NOP[UBXREF]

A public cross-reference list file is not output.

(2) PUB[XREF]

A public cross-reference list file is output.

(3) Designation of the filename.

The output filename may be designated after PUB[XREF]. The entry format for the filename should be "[drive name: [/directory/]]" filename. It is also possible to enter AUX, CON, PRN and NUL in the filename. If an output filename is not entered, the directory and filename output will be as follows:

- 1 If using a sequence file  
The sequence file and the same directory.  
The sequence file and the same filename with the extension .PUB.
- 2 If there are two or more source files to be assembled (a sequence file is not being used)  
The device file and the file of the same name with the extension .PUB  
The source file name and the same directory  
The extension may be omitted from the entry when designating output filenames. In this situation, .PUB will automatically be assigned as the output file extension.

(4) [, PRN]

If ", PRN" is entered following the filename designation, the public cross-reference list file output will also go to the printer. However, if PUB=PRN or PRN is entered, an error will be designated.

- (5) If there is one source file only to be assembled, this option will be disregarded.

[Default value]

NOB[UBXREF]

## 4.5.20 The operational drive control option

[Entry format]

```
[ WORK[K]=Drive name: ]
```

[Function]

This option designates the drive name which holds the working files used in assembling.

[Explanation]

(1) Drive name specification

Only one drive name may be specified.

[Example]

WORK=A:

(2) After assembling is complete, all working files are deleted.

[Default value]

Current drive.



### 4.5.21 The program summary output option

[Entry format]

```
[ SUMMARY="Title" [,Filename] ]
```

[Function]

This option designates a certain filename for entry of a program summary, and designates output of its contents to a document list file. A part of a character string enclosed in double quotation marks will be output at the start of the program summary as its title.

[Explanation]

- (1) A maximum of 255 characters may be enclosed by the double quotation marks.
- (2) If an assemble option is designated in the command line, or spaces are used as the character string in the title section, an error will be generated and assembling will halt.
- (3) This option will not be operative with document list file output option designation /NOD.

[Default value]

Output not designated.

## UMAS17K ASSEMBLER

---

### 4.5.22 The SIMPLEHOST data output control option

[Entry format]

[ { HOS[T] } ]
[ { NOH[OST] } ]

[Function]

This option control whether or not to output data required when using the uPD17000 series development tool SIMPLEHOST.

[Explanation]

(1) NOH[OST]

SIMPLEHOST data is not output.

(2) HOS[T]

SIMPLEHOST data is output to a .OBJ file.

(3) If HOS[T] is selected, the following assemble option is forced:

```
/LIST/HEX/COND/GEN
```

In addition to this, object codes output to the assemble list are all in C4444 format, and C14344 and SFCOND control instructions entered in the source are inoperative.

[Default value]

NOH[OST]

### CHAPTER 5 ASSEMBLER OUTPUT LISTS

#### 5.1 Types of Output Lists

The AS17K is provided with the capacity to output the following lists after assembling.

Output list	Output file extension	Assemble option	Module designated
Assemble list	.PRN	/LIS[T]	o
Option information list	.PRN	/SEQ	o
Cross-reference lists	.XRF .PRN	/XRE[F]	o
Memory maps	.MAP	/MAP	
Public cross-reference lists	.PUB	/PUB[XREF]	
Assemble reports	.REP	/REP[ORT]	
Documents	.DOC	/DOC[UMENT]	

If it is desired to output one of the lists as illustrated above, it is necessary to designate that list with an assemble option when assembling starts. For the method of designation, please refer to Section 4.5 on the assemble options. If no designation is made with an option, an assemble list, an option information list, and a cross-reference list will be output to the directory in the source file (.ASM) as a source file (.PRN) file.

If it is not desired to output these lists, "NO" should be appended before the option name:

(example:/NOLIST/NOSEQ...).

Apart from the above, the following files may be output.

(1) A logging data file (AS17K.LOG)

This file automatically records start-up messages, begin messages, error messages and end messages output to the console from the point at which assembling begins to its conclusion.

(2) Object files (HEX or PROM files)

As with the other lists, these may be designated by an output control option.

**5.2 List Output Format Controls**

- (1) Number of lines per page: This is set in accordance with the assemble option "ROW=n" (n must be greater than 50 but less than 250).
- (2) The number of columns per line: This is set in accordance with the assemble option "COL=n" (n should be greater than 72 but less than 132).

If more than the permissible number of columns in a line is designated, the section in excess will be deleted. If there are full-em characters at the position from which the deletion is to occur, the deletion will take place from one column prior to that point.

- (3) Page break control: This is set in accordance with the assemble option "FORM/NOFORM".

FORM: Sends an FF/LF code to insert a page break in a list.

NOFORM: Outputs carriage return or line feed codes up to the line designated by the ROW option.

Note: FF (the form feed code) is the code 0CH.  
LF (the line feed code) is the code 0AH.  
CR (carriage return) is the code 0DH.

- (4) TAB code control: This is set in accordance with the assemble option [TAB/NOTAB].

NOTAB: When lists are output, spaces are inserted at TAB code positions in the same way that characters coming after TAB codes appear in each eighth column after the beginning of the line.

TAB: When lists are output, TAB codes are output as they are.

### 5.3 Outputting the Header

In lists other than documents, the following headers are output as the first paragraph on each page.

- (1) The assembler name and the version number of the assembler.
- (2) The device filename and the version number of the device file.
- (3) The listing title.
- (4) The time of assembling, and the page (module order number and page number within the module).
- (5) The program name: The character string designated by the PROG option.
- (6) The module name.

Example: UPD17000.ASM

#### 5.4 Option Data Lists

Data may be output on assemble options, device filenames and source filenames designated when assembling starts.

(1) Output control options

/SEQ, /NOSEQ

However, note that if NOLIST has been set, a sequence data file will not be output even though SEQ is designated.

(2) Output file names

These are output on page one of the assemble list.

/LIST: sourcefile. PRN

/LIST=filename: .PRN

(3) Data output

① The assemble option is output from the command line.

Example: COMMAND LINE OPTION=D17102.DEV/SEQ/LIS 17102.ASM/XRE

② If a sequence file has been designated, the sequence filename and the contents of the file will be output.

Example: SEQ FILE=B:¥17102.SEQ

D17102.DEV

/SEQ/LIS/COL=100/PROG="17K SAMPLE"

MOD1.ASM

MOD2.ASM/XRE

MOD3.ASM

③ The module option display

Effective assemble options and their modules may be output if designated in sequence files and command lines when beginning assembling. If no option column is designated, the value will be the default value. If a character string is not designated with the SUMMARY or PROGRAM options, nothing in that column will be output.

<Output Example>

AS17K E1.OK 07 «μPD17051 ASSEMBLE LIST» 08:31:00 05/24/88 PAGE 01-001

PROG = SI0 MODE

COMMAND LINE OPTION =

SEQ FILE = SI0.SEQ

A:\$AS17K\$μPD17051.DEV  
/DOC/MAP/FOR/NOT/  
/COL=120/PROG="SI0 MODE"  
SI0MST.ASM  
SI0SLV.ASM

<MODULE OPTION>

NOOBJECT/OBJECT	:	OBJECT=SI0MST.OBJ
NOHEX/HEX	:	HEX=A:\$AS17K\$μPD17051.HEX
NOPROM/PROM	:	NOPROM
NOLIST/LIST	:	LIST=SI0MST.PRN
NOXREF/XREF	:	XREF=SI0MST.PRN
NOERROR/ERROR	:	NOERROR
ROW	:	ROW=66
NOGEN/GEN	:	GEN
COLUMN	:	COLUMN/=120
NOAUTOLOAD/AUTOLOAD	:	NOAUTOLOAD
NOCOND/COND	:	COND
NOSEQ/SEQ	:	SEQ
NOMAP/MAP	:	MAP=SI0.MAP
NODOCUMENT/DOCUMENT	:	DOCUMENT=SI0.DOC
NOBRANCH/BRANCH	:	NOBRANCH
PROG	:	PROG=SI0 MODE
NOTAB/TAB	:	NOTAB
NOFORM/Form	:	FORM
NOREPORT/REPORT	:	NOREPORT
NO PUBXREF/PUBXREF	:	NO PUBXREF
SUMMARY	:	SUMMARY=
WORK	:	WORK=
ZZZ0	:	ZZZ0=0
ZZZ1	:	ZZZ1=0
ZZZ2	:	ZZZ2=0
ZZZ3	:	ZZZ3=0
ZZZ4	:	ZZZ4=0
ZZZ5	:	ZZZ5=0
ZZZ6	:	ZZZ6=0
ZZZ7	:	ZZZ7=0
ZZZ8	:	ZZZ8=0
ZZZ9	:	ZZZ9=0
NOHOST/HOST	:	NOHOST

### 5.5 Assemble Lists

It is possible to output statements and object codes, and so forth in source programs.

(1) Output control options

/LIST,/NOLIST

(2) Output file names

/LIST:SOURCE FILE.PRN

/LIST=filename: filename.PRN

(3) Output formats

① Line format

The line header format is as set out below.

E STNO LOC. OBJ. M I SOURCE STATEMENT

E: Error code (a single alphanumeric character indicates the type of error).

STNO: Line number (corresponds to the source file line numbering).

LOC: Location address (program memory address).

OBJ.: Object code.

M: Macro nesting level.

I: Include nesting level.

SOURCE STATEMENT: Details of the source programs.

② Outputting development statements from the `acro/repeat/include` sections.

If macro, repeat or include sections are developed, output is as set out below.

- o The STNO field may be output with a + sign and the development statement line number until the point at which macro development terminates.

The development statement line number starts from +1 for each development.

- o The macro and iteration directive nesting level may be output for the M field.
- o The include directive nesting level may be output for the I field.



③ Outputting error statements

Statements in which errors have been generated are output with the error code at the beginning of the line, and the error number at the end of the line. For error numbers and codes, please refer to the error and warning messages in Chapter 6.

④ Object output formats

Object codes in control instructions may be output in the following formats.

o C14344

o C4444

For further details, please refer to Section 3.3 on control instructions in Part 1.

## UMAS17K ASSEMBLER

<Output Example>

AS17K E1.OK 07 «μPD17051 ASSEMBLE LIST» 08:31:00 05/24/88 PAGE 01-003

PROG = SIO MODE

SOURCE = S10MST.ASM

```

E STNO LOC. OBJ. M 1 SOURCE STATEMENT
+ 5 2 CLR1 SIOCH
+ 6 0012 07038 2 PEEK WR.ME.SIOCH SHR 4
+ 7 0013 14787 2 AND WR.#.DF.(NOT SIOCH) AND 0FH
+ 8 0014 07028 2 POKE .MF.SIOCH SHR 4.WR
43 1 INITFLG NOT SBACK.SIONWT.SIOWR01 NOT SIOWRQ0
:SBACK<-0.SWT<-1.SWRQ1<-1.SWRQ0<-0
+ 1 2 SET2 SIONWT.SIOWRQ1
+ 2 0015 07138 2 PEEK WR.MF.SIONWT SHR 4
+ 3 0016 16786 2 OR WR.#.DF.(SIONWT OR SIOWRQ1) AND 0FH
+ 4 0017 07128 2 POKE .MF.SIONWT SHR 4.WR
+ 5 2 CLR2 SBACK.SIOWRQ0
+ 6 0018 07138 2 PEEK WR.MF.SBACK SHR 4
+ 7 0019 14786 2 AND WR.#.DF.(NOT (SBACK OR SIOWRQ0) AND 0FH)
+ 8 001A 07128 2 POKE .MF.SBACK SHR 4.WR
44 ;SLAVE ADDR TRANSFER START!
45 CLR2 POAB100.POABI01 ;SPECIFY POA0 & POA1 INPUT PORT
+ 1 001B 07337 1 PEEK WR.MF.POAB100 SHR 4
+ 2 001C 1478C 1 AND WR.#.DF.(NOT (POAB00 OR POAB101) AND 0FH)
+ 3 001D 07327 1 POKE .MF.POAB100 SHR 4.WR
48 SKF1 RXFLG ;1F RX
+ 1 001E 1F104 1 SKF .MF.RXFLG SHR 4.#.DF.RXFLEG. AND 0FH
47 001F 0C02E BR CHXACK ;THEN GOTO CHXACK
48 :*****
49 :. 'DATA TRANSFER'
50 :*****
51 :
52 DATA.TX:
53 0020 0B510 SKNE DCNT. #0 ;IF ALL THE DATA HAS BEEN TRANSFERRED.
54 0021 0C054 BR STOP-COND ;THEN GOTO STOP CONDITION PROCESS
55 SET1 IXE ;INDEX MODE START
+ 1 0022 167F1 1 OR .MF.IXE SHR 4.#.DF.IXE AND 0FH
56 0023 0850F LD DBF0.1XADDR ;MOVE (MEM) ADDRESSED BY 1X TO DBF0
57 0024 07080 INC 1X ;INCREMENT INDEX REGISTERS
58 0025 0850E LD DBF1.1XADDR ;
59 0026 0A7080 INC 1X
60 CLR1 1XE ;INDEX MODE END
+ 1 0027 147FE 1 AND .MF.1XE SHR 4.#.DF.(NOT 1XE AND 0FH)
61 0028 11511 SUB DCNT. #1 ;DCNT=DCNT-1
62 TXB8:
63 SKT1 SIOSF8 ;WAIT FOR RISING EDGE OF
+ 1 0029 07238 1 PEEK WR.MF.SIOSF8 SHR 4
+ 2 002A 1E788 1 SKT WR.#.DF.SIOSF8 AND 0FH
64 002B 0C029 BR TXB8 ;SHIFT CLOCK FOR BIT#8 OR #9
65 002C 070A3 PUT SIOSFR.DBF ;SIOSFR <- DATA TO SEND

```

### 5.6 Cross-reference Lists

Cross-reference lists are lists which output symbol types used in a source program, evaluations, and the numbers of lines containing definitions or references.

#### (1) Output control options

/XREF, /NOXREF

#### (2) Output filenames

/XREF: sourcefile. PRN (however, the source file extension is XRF when NOLIST is designated).

/XREF=filename: filename .PRN (XRF)

#### (3) Data output

##### ① Line format

The line header format is as set out below.

SYMBOL TYPE A VALUE REF (#DEF)

SYMBOL: Symbol name.

Symbol names are sorted in shift JIS code order (with en designations, no distinction is made between upper case and lower case characters), and output in up to 16 en characters or 8 characters.

TYPE: Symbol type

Data type : DAT

Flag type : FLG

Memory type: MEM

Label type : LAB

Macro : MAC

Others : Flag display

A: Symbol attributes

PUBLIC : P

LOCAL : L

EXTRN : E

VALUE: Symbol value

DAT : <evaluation>

LAB : <evaluation>

MEM : <bank number>. <address>

FLG : <bank>. <address>. <bit position>

MAC : \*\*\*\*\* (no symbol value display)

- o However, with MEM and FLG types, an evaluation will be output if entry has been effected without the use of a bit position segment indicator.
- o Symbols defined in other modules may be output to this column with "EXTRN".

REF (#DEF): Reference line number (#definition line number)

If the # is attached, the number is a definition line number.



### 5.7 Memory Maps

MEM and FLG type symbols used in source programs are output in memory map format. Subsequent to this, the symbols can be output in list format with more detailed data.

(1) Output control options

/MAP, /NOMAP

(2) Output filenames

/MAP: Source file.MAP

/MAP=filename: Filename.MAP

(3) Structure

Memory maps may be output in the following order in the one file, which will have the extension .MAP. If the source program is split into modules, module units may be output in the format given below. The output order is the assemble order.

Memory map

Flag map

Symbol list

- o After each bank has been output or if, during output, the effective number of lines as designated by ROW is reached, a page break will be inserted.
- o In situations in which one address line cannot be output on the page, or it is not possible to output only one page, a page break will be inserted immediately prior.
- o Symbol names may be output in up to eight en characters. If this number is exceeded, an en asterisk will be substituted for the eighth een character and printed. If four em characters is exceeded, an en asterisk will be printed in place of the fourth character.

### 5.7.1 Memory maps

Each bank may be output as described below.

- o Address columns which contain symbols are divided into upper and lower portions by a row of dots. The names of symbols whose attributes are PUBLIC will appear in the top section, while symbols with LOCAL attributes will appear in the lower section.
- o It is possible to output up to 67 symbols in the one address. Any more symbols will be disregarded.
- o If there is no MEM type symbol in the source program, there will be no output.

## UMAS17K ASSEMBLER

<Output example>

AS17K E1.0K 07 << UPD17051 DATA MEMORY MAP >> 10:29:59 05/23/88 PAGE 02-001

PROG =

SOURCE = I2CSLY.ASM

(PUBLIC/LOCAL MAP)

BANK = 0

ROW

7	6	5	4	3	2	1	0	
		!SAVEWR						0
		!XADDR						
		!SAVERPL		!SIO13	!SIO12			1
		!DCNT						
								2
		!SAVEDBF						3
								C
		!SAVEPSW						4
								L
		!SAVEDPH						5
		!SAVEPL						6
								7
								8
								9
								A
								B
								C
								D
								E
								F



### 5.7.2 Flag maps

Each bank may be output as shown below.

- o Data memory addresses which have symbols in the flag map left column, or upper part of the column, will output bit position numbers.
- o Address columns which contain symbols will be divided into upper and lower sections by a line of dots. Symbols with PUBLIC attributes will appear in the top section while LOCAL attribute symbols will appear in the lower section.
- o If there is no FLG type symbol, there will be no output.
- o Symbol names are output in up to eight characters. If this number is exceeded, an asterisk will be printed in the place of the eighth character. If four characters is exceeded, an en asterisk will be printed in the place of the fourth character.

<Output example>

AS17K E1.0K 07 << UPD17051 FLG MAP >> 10:29:59 05/23/88 PAGE 02-002

PRCG =

SOURCE = IZCSLV.ASM

(FLAG MAP) BANK = 0

MSB	3	2	1	.0	LSB
10	.....	.....	!	!	!
	!SIOERR	!RFLG	!	!	!
OF	!	!	!	!	.....
	!	!	!	!	!DBF00

**5.7.3 Symbol lists**

Symbols displayed in memory maps and flag maps may be output as follows:

The line header will be as given below.

**SYMBOL TYPE BNK LOC BIT INFORMATION**

**SYMBOL:** The symbol name is output in up to eight en characters. If the symbol name exceeds eight en characters, an asterisk will be printed out at the eighth character. If four characters is exceeded, an asterisk will be printed out at the fourth character.

**TYPE:** The symbol type should be either MEM or FLG.

**BNK:** Bank number

**LOG:** Data memory address

**BIT:** The bit is a FLG type symbol only whose position may be output. The relevant bit of 4-bits is displayed as 1, while other bits are displayed as (.).

**INFORMATION:** Comments in lines which define symbols in source programs may be output.

<Output example>

AS17K E1.0K 07 << UPD17051 DATA MEMORY MAP >> 10:29:59 05/23/88 PAGE 02-003

PROG =

SOURCE = I2CSLV.ASM

(DATA AREA INFORMATION)

SYMBOL	TYPE	BANK	LOC	BIT	INFORMATION
DBF00	FLG	0	F	...1	
SAVEDBF	MEM	0	53		SAVE AREA OF DBF-0
SAVEMPH	MEM	0	55		SAVE AREA OF MPH
SAVEMPL	MEM	0	56		SAVE AREA OF MPL
SAVEPSW	MEM	0	54		SAVE AREA OF PSW
SAVERPL	MEM	0	51		SAVE AREA OF RPL
SAVEWR	MEM	0	50		SAVE AREA OF WR
S1012	MEM	0	12		S10 MODE SELECT
S1013	MEM	0	13		S10 MODE SELECT
DCNT	MEM	0	51		
IXADDR	MEM	0	50		
FXFLG	FLG	0	10	...1	
SIOERR	FLG	0	10	1...1	DEFINE BIT#3 IN DATA MEMORY ADDR 10H OF BANK 0

### 5.8 Assemble Reports

These reports contain the results of executing assemble operations.

(1) Output control options

    /REP, /NOREP

(2) Output filenames

    /REP:sourcefile.REP

    /REP=filename: filename.REP

(3) Types of reports

The following types of assemble reports may be output.

    Module reports

    Final phase reports

    Total reports

#### 5.8.1 Module reports

Module reports are lists which output the time required for assembling each source module, the memory area required for assembling, and the file sizes, in bytes of both input and output files.

(1) Time required for assembling

A list may be output containing the assemble start time, finish time and time elapsed.

(2) Memory volume used in assembling

The assemble area is the volume of memory provided when assembling starts. If the volume of this region is exceeded during assembly, an error will be generated.

(3) Input and output file sizes

o As with assemble and cross-reference lists, files with the extension .PRN may be output: only the size of the .PRN file will be displayed.

o The sizes of input and output files used in assembling may be output as is shown in the following examples. However, temporary files may not be output.

<Output example>

AS17K E1.0K 06 << UP017002 REPORT >>

09:16:38 05/23/88 PAGE 02-001

PROG =

SOURCE = SUNCK2.ASM

< MODULE REPORT >

PROCESS	START	END	ELAPSED
ASSEMBLE PHASE - 1	09:16:21	09:16:22	00:00:01
ASSEMBLE PHASE - 2 (PASS 1)	09:16:22	09:16:27	00:00:05
ASSEMBLE PHASE - 3 (PASS 2)	09:16:27	09:16:27	00:00:00
PROGRAM LISTING	09:16:27	09:16:38	00:00:11
CROSS REFERENCE LISTING	09:16:35	09:16:35	00:00:00
MAP LIST	09:16:35	09:16:35	00:00:00
		TOTAL	00:00:17

AREA NAME	SIZE	USED	
SYSTEM MEMORIES	174176	65107	37 X
DEVICE FILE TABLE ( RESERVED WORKS...etc )	6193	6193	100 X
INTERMEDIATE CODE	22751	732	3 X
INTERMEDIATE CODE FILE	22751	0	0 X
SYMBOL TABLE	49620	2355	4 X
CROSS REFERENCE TABLE	11390	1494	13 X
MACRO TABLE	11390	0	0 X
	0	0	0 X
SOURCE BUFFER	22780	5567	24 X
FILE I/O BUFFER	1024	1024	100 X
OTHER AREA	38422	20162	52 X

## UMAS17K ASSEMBLER

---

<Output example>

AS17K E1.0K 06 << UP017002 REPORT >>

09:18:38 05/23/88 PAGE 02-002

PROG -

SOURCE = SUNCK2.ASH

< MODULE REPORT >

	FILE NAME	PAGE	SIZE	LINE
SOURCE	SUNCK2.ASH		5567	330
INCLUDE				
OBJECT				
PRN LST	SUNCK2.PRN	6	15128	392
XRF LST				
HEX MAP	SUNCK01.MAP	3	8369	186
WORK				

TOTAL ERRORS = 13

TOTAL WARNINGS = 0

### 5.8.2 Final phase reports

The time required for the final phase of each process, and the file sizes, may be output as shown below.

A final phase report is output giving the time required for link processing after source modules have been assembled, the time required to generate the output files created through link processing, the time required for the generation of document files, and the amount of memory used.

<Output example>

AS17K E1.0K 06 << UPD17002 REPORT >>                    09:17:00 05/23/88                    PAGE 00

PROG =

< FINAL PHASE REPORT >

PROCESS	START	END	ELAPSED
LINK	09:16:39	09:16:40	00:00:01
PUBLIC CROSS REFERENCE	09:16:39	09:16:39	00:00:00
DOCUMENT LIST	09:16:40	09:17:00	00:00:20
		TOTAL	00:00:21

FILE NAME	SIZE
HEX	
PROH	
PUB LIST	
DOC LIST	SUMCKQL.DOC
	7229

5.8.3 Total reports

Total reports list the total time required for all assembly operations from start to finish, the number of public symbols, the number of local symbols, the number of macro developments, the number of lines, the total number of errors and the total number of warnings.

<Output example>

AS17K E1.DK 06 << UPD17002 REPORT >> 09:17:00 05/23/88 PAGE 00

PROG -

< TOTAL REPORT >

PROCESS	START	END	ELAPSED
START PHASE	09:16:07	09:16:09	00:00:02
SUMCK1.ASH	09:16:09	09:16:21	00:00:12
SUMCK2.ASH	09:16:21	09:16:39	00:00:18
FINAL PHASE	09:16:39	09:17:00	00:00:21
		TOTAL	00:00:53

PUBLIC SYMBOLS = 162

LOCAL SYMBOLS = 1

MACRO EXPANTION — 0 TIMES, 0 LINES

TOTAL ERRORS = 14

TOTAL WARNINGS = 1

END OF REPORT



### 5.9 Public Cross-reference Lists

This type of list outputs the symbol cross-references created through referring to external modules when carrying out split assembling.

(1) Output control options

/PUB/NOPUB.

(2) Output filenames

/PUB: sourcefile.PUB

/PUB=filename: filename.PUB

(3) Data output

① Line format

The line header is as set out below:

SYMBOL TYPE VALUE REF (#DEF)

SYMBOL: Symbol names are sorted in shift JIS code order (no distinction is made between upper case and lower case characters when they are designated as en characters) and up to 16 en or 8 characters may be output.

TYPE: The symbol type may be output

VALUE: The symbol value

REF (#DEF): The line number of the reference (# indicates the line number of a definition)

The output format for line numbers is [mm]

llll--iii

mm : module number

llll: line number in a module

iii : development line number

If one reference or definition line number only can be output, consecutive lines will be output.

<Output Example>

AS17K E1.OK.05 «μPD17102 PUBXREF LIST» 09:16:01 05/23/88 PAGE 001

PROG =

SYMBOL	TYPE	VALUE /REF (#DEF)
AKEY51	MEM	0.08 / [1]# 17 . [2] 14
AKEY52	MEM	0.0C / [1]# 18 . [2] 15
AKEY53	MEM	0.0D / [1]# 19 . [2] 16
AUTO01	FLG	0.63.1 / [1]# 20 . [2] 17
AUTO0F	FLG	0.64.2 / [1]# 21 . [2] 18
FLG1	FLG	0.10.1 / [1] 7 . [1]# 12 . [2] 7
KEYI	DAT	E / [1]# 27 . [2] 24
KEYJ	DAT	1 / [1]# 26 . [2] 23
KEYP	DAT	4 / [1]# 28 . [2] 25
MEM1	MEM	0.01 / [1] 6 . [1]# 10 . [2] 6
MEM2	MEM	0.02 / [1] 6 . [1]# 11 . [2] 6
LABEL1	MEM	1.33 / [1]# 22 . [2] 19
LABEL2	MEM	1.34 / [1]# 23 . [2] 20
LABEL3	MEM	1.35 / [1]# 24 . [2] 21
LABELA	LAB	200 / [1]# 30 . [2] 27
LABELB	LAB	300 / [1]# 31 . [2] 28
LABELC	LAB	400 / [1]# 32 . [2] 29

TOTAL SYMBOLS = 17

END OF PUBLIC XREF LIST

### 5.10 Documents

Documents consist of text and tables of contents.

#### (1) Output control options

/DOC, /NODOC

#### (2) Output filenames

/DOC: Sourcefile.DOC

/DOC=filename: filename.DOC

### 5.10.1 Tables of contents

Tables of contents to documents display each title in the text of the document and its page number.

[Example]

ASI7K E1. OK 07 «MPD17051 DOCUMENT» 10:30:07 05/23/88

TABLE OF CONTENTS		PAGE	
SIO MASTER MODULE	←	1	
SIO MASTER MODE EXAMPLE	}	1	
,DEFINITION OF FLAG'		1	
,DEFINITION OF MEMORY'		1	
,START CONDITION'		1	
,SEND SLAVE ADDR'			
,DATA TRANSFER'			
,ACK CHECK'			
,CHECK DATA DIRECTION'			
,DATA RECEIVE'			
,STOP CONDITION			
SIO SLAVE MODULE			
SIO SLAVE MODE EXAMPLE		Designated with the SUMMARY control instruction	4
,DEFINITION OF MEMORY'			4
,ENABLE SIO SLAVE'			
,DEFINITION OF SAVE AREA'			
,SAVE SYSTEM REGISTERS PART I'		5	
,SLAVE ADDR COMPARE'		5	
,SAVE SYSTEM REGISTERS PART II'			
,JUDGE READ OR WRITE'			
,SLAVE TX PROCESS'			
,SLAVE RX PROCESS'			
,TERMINATE PROCESS'			

- o The title "TABLE OF CONTENTS" is output to page 1 only.
- o The line spacing is set by the SUMMARY control instruction line spacing command. LFnn. The default is single line spacing.
- o To end the table of contents, a space is inserted when entering the title with the SUMMARY control instruction or the program summary output control option.

### 5.10.2 The text of the document

There are three levels in the text of a document: programs, modules and routines corresponding to the source program. Programs, modules and routines are output as "titles", "summaries", and "data".

Program titles and summaries are entered with the program summary output control instruction, while module and routine titles and summaries are entered with the SUMMARY control instruction. Data on symbols in modules and routines can be automatically output. For more details, please refer to Part 1 Section 3.5 on the documentation generation functions.

- (1) The title of a program is a character string designated by the first parameter in the program summary output instruction option.
- (2) The program summary is the contents of the file designated by the second operand in the program summary output control option.

```
/SUM[MARY]="0.0 ABSTRACT"
```

A filename summary is a character string entered in a file designated by a filename. If there is no file, no program summary text will be output.

- (3) Module titles and summaries

Titles and summaries may be designated with the SUMMARY control instruction which appears at the beginning of each module and output.

- (4) Module data

- (a) Lists PUBLIC symbols declared in modules.
- (b) Lists EXTRN symbols declared in modules.
- (c) Program memory address ranges in modules.

With (a) and (b), the symbol name is output distinguished by type. The character string enclosed in parentheses following PUBLIC or EXTRN is the symbol type. (c) is output in four hexadecimal digits.

- (5) Routine titles and summaries

Routine titles and summaries are designated by second and subsequent SUMMARY directives appearing in each module.

(6) Routine data -- Symbol lists in summaries

Symbols referenced in summaries of routines are sorted by symbol name and output. If a symbol is referenced several times, it will still be output only once. For the meanings of symbols output, please refer to the section on the documentation generation control instruction.

- o The effective number of characters in a symbol name is 12 en characters; characters in excess of 12 will not be output.
- o After symbols which are output to the "BRANCH TO" page, the line in which that symbol is entered in the source program, and comment statements from the same line, will be output.
- o If parentheses are attached to a symbol name, it indicates that that symbol is used in an operation.

(7) Titles

Titles are output as character strings enclosed in quotation marks following the command .TITLE, which is used by the SUMMARY control instruction to enter titles.

<Source list input example>

```

SUMMARY '$, 'SIO MASTER MODULE'

This module gives an example of SIO interface master mode.
$
SUMMARY % 'SIO MASTER MODE EXAMPLE.

Gives an example of SIO master transmission and reception modes.
%
:*****
:. 'DEFINITION OF FLAG'
:*****
:
SIOERR  FLG      0.10H.3 ;DEFINE BIT#3 IN DATA MEMORY ADDR OF 10H BANK 0
                          ;AS SIO ERROR FLAG
POA1    FLG      0.70H.1 ;DEFINE BIT#1 IN DATA MEMORY ADDR 70H OF BANK 0
POA0    FLG      0.70H.0 ;DEFINE BIT#0 IN DATA MEMORY ADDR 70H OF BANK 0
RXFLG   FLG      0.10H.2
DBF00   FLG      0.0FH.0
    
```

<Assemble list output example>

```

AS17K E1.OK.07 «µPD17051 ASSEMBLE LIST» 08:31:00 05/24/88 PAGE 01-002

PROG = SIO MODE

SOURCE = SIO MST.ASM

E STNO  LOC.  OBJ.  M I SOURCE STATEMENT
  1
  2          SUMMARY '$, 'SIO MASTER MODULE'
  3
  4          This module gives an example of SIO interface master mode
  5          $
  6          SUMMARY % 'SIO MASTER MODE EXAMPLE.
  7
  8          Gives an example of SIO master transmission and reception
  9          modes.
 10
 11          %
 12          :*****
 13          :. 'DEFINITION OF FLAG'
 14          :*****
 15          :
 16          SIOERR FLG 0.10H.3 ;DEFINE BIT#3 IN DATA MEMORY ADDR 10H OF BANK 0
 17          ;AS SIO ERROR FLAG
R 15 0000 074F0 POA1  FLG 0.70H.1 ;DEFINE BIT#1 IN DATA MEMORY ADDR 70H OF BANK 0 049
R 16 0002 074F0 POA0  FLG 0.70H.0 ;DEFINE BIT#0 IN DATA MEMORY ADDR 70H OF BANK 0 049
 17          0104  RXFLG  FLG 0.10H.2
 18          00F1  DBF00  FLG 0.0FH.0
 19          :
    
```

<Output example>

Page 1

SI0 MASTER MODULE

This module gives an example of SI0 interface master mode.

ADDR RANGE : 0000H - 0021H

SI0 Master mode example.

Gives an example of SI0 master transmission and reception modes.

ENTRANCES	:-	
MEMORIES CHANGED	:SAVEDBF	SAVEMPH
	SAVEMPL	SAVEPSW
MEMORIES REFERRED	:-	
MEMORIES MANIPULATED	:DCNT	
FLAGS CHANGED	:RFFLG	
FLAGS REFERRED	:-	
DATA REFERRED	:-	
BRANCH TO	:SI01SET	
SUBROUTINES CALLED	:-	
LABELS MANIPULATED	:-	
SYSTEM CALL	:-	





### CHAPTER 6 ERROR AND WARNING MESSAGES

#### 6.1 Assembling Errors

If the AS17K detects errors in parameter entries designated when assembling, an error message will be displayed and assembling will halt.

	Message	file not found
	Cause	File designated not in drive and directory designated
	Program action	Assembling terminates
	User action	Designate the correct file
	Message	File length failed
	Cause	Data required to start assembling is not in the file designated.
	Program action	Assembling terminates
	User action	Designate the correct file
	Message	file too large
	Cause	The volume of the file designated is too large for the memory
	Program action	Assembling terminates
	User action	Increase memory volume or reduce the size of the file
	Message	Invalid file extension name
	Cause	Filename designated following the device filename [.DEV] does not have the extension .DEV
	Program action	Outputs a prompt for a device filename again
	User action	Input the correct device filename
	Message	Invalid option
	Cause	Incorrect option setting -- option name or parameters are incorrect
	Program action	Outputs incorrect option and terminates assembling
	User action	Specify the correct option

Message	Out of memory
Cause	Memory volume insufficient
Program action	Assembling terminates
User action	Reduce the number of options, or increase memory volume, or change the working drive designation

## 6.2 Errors which relate to source programs

If there is an error in an entry in a source program, the line number and statement containing the error will be output to the monitor along with an error message when assembling is executed. In addition to this, an error code will be printed at the beginning of the statement line in the corresponding assemble list, with an error number at the end of the line. The messages displayed on the monitor are all stored in a file called AS17K.LOG; it is therefore possible to check them later. Warning messages (Code: W) do not appear on the monitor, but are stored in the AS17K.LOG file. When an error has been generated, the assembler ignores that line and continues assembling. However, if it is determined that the line containing the error also contains a uPD17000 instruction, NOP (074F0) is assigned as an object code.

No.11	Code O	Message	Illegal first operand type
		Cause	The first operand type is illegal.
		User action	Enter the correct type expression.
No.12	Code O	Message	Illegal second operand type
		Cause	The second operand type is illegal.
		User action	Enter the correct type expression.
No.13	Code O	Message	Illegal third operand type
		Cause	The third operand type is illegal.
		User action	Enter the correct type expression.
No.14	Code V	Message	Illegal first operand value
		Cause	The first operand value is illegal.
		User action	Check that the operand value is permitted by the product.
No.15	Code V	Message	Illegal second operand value
		Cause	The second operand value is illegal.
		User action	Check that the value of the operand is permitted by the product.
No.16	Code V	Message	Illegal third operand value
		Cause	The third operand value is illegal.
		User action	Check that the value of the operand is permitted by the product.
No.17	Code S	Message	Must be comma
		Cause	A comma has not been entered.
		User action	Enter a comma in the correct position
No.18	Code R	Message	Out of address range
		Cause	The address range is incorrect.
		User action	Check the program memory address value, and enter a correct value.
No.19	Code A	Message	Illegal addressing
		Cause	The addressing operation is incorrect.
		User action	Perform the address operation correctly

No.20	Code W	Message	Unreferenced symbol
		Cause	The symbol has not been referenced.
		User action	Check if the symbol is necessary. If it is not, delete it; if it is, make a reference to it.
No.21	Code P	Message	No IF statement
		Cause	No IF statement to correspond to an ENDIF.
		User action	Enter the IF statement in the correct position.
No.22	Code P	Message	No CASE statement
		Cause	No CASE statement corresponding to an ENDCASE.
		User action	Enter the CASE statement in the correct position.
No.23	Code P	Message	No REPT statement
		Cause	No REPT statement corresponding to an ENDR.
		User action	Enter the REPT statement in the correct position.
No.24	Code P	Message	No IRP statement
		Cause	No IRP statement corresponding to an ENDR.
		User action	Enter an IRP statement in the correct position.
No.25	Code S	Message	Symbol define error
		Cause	Symbol definition is incorrect.
		User action	Enter the symbol definition directive and the operand correctly.
No.26	Code A	Message	Invalid address
		Cause	The address specification is incorrect.
		User action	Enter the correct address.
No.27	Code P	Message	No OPTION statement
		Cause	No OPTION statement corresponding to an ENDOP.
		User action	Enter an OPTION statement in the correct position.

No.28	Code P	Message	No END statement
		Cause	No END at the end of a statement.
		User action	Enter the END statement
No.29	Code P	Message	No ENDIF statement
		Cause	No ENDIF statement for an IF statement.
		User action	Enter an ENDIF statement in the correct position
No.30	Code P	Message	No ENDCASE statement
		Cause	No ENDCASE statement for a CASE statement.
		User action	Enter a ENDCASE statement in the correct position.
No.31	Code P	Message	No ENDR statement
		Cause	No ENDR statement for REPT or IRP.
		User action	Enter an ENDR statement in the correct position.
No.32	Code P	Message	No ENDM statement
		Cause	No ENDM statement for a MACRO.
		User action	Enter an ENDM statement in the correct position.
No.33	Code P	Message	No ENDP statement
		Cause	No ENDP statement corresponding to PUBLIC BELOW.
		User action	Enter an EDP statement in the correct position.
No.34	Code P	Message	No ENDOP statement
		Cause	No ENDOP statement for OPTION.
		User action	Enter an ENDOP statement in the correct position.
No.35	Code N	Message	Nesting overflow
		Cause	40 levels of nesting exceeded with IF, MACRO, REPT,IRP, etc.
		User action	Reduce nesting levels to 40 or below.
No.36	Code O	Message	Operand count error
		Cause	Number of the operand is incorrect.
		User action	Enter the correct number of operands.

No.37	Code S	Message	Syntax error
		Cause	Syntax error.
		User action	Enter using the correct syntax.
No.38	Code M	Message	Syntax memory overflow
		Cause	System memory is insufficient.
		User action	Increase the memory area.
No.39	Code S	Message	Symbol area overflow
		Cause	Symbol area is insufficient.
		User action	Increase the symbol area or decrease the number of symbols.
No.40	Code P	Message	Invalid EOF statement
		Cause	Incorrect EOF statement is described.
		User action	Delete if not necessary.
No.41	Code P	Message	Invalid ENDR statement
		Cause	ENDR statement in wrong position.
		User action	Enter the statement in the correct position.
No.42	Code P	Message	Invalid EXITR statement
		Cause	EXITR entered in the wrong position.
		User action	Enter EXITR In the correct position.
No.43	Code P	Message	Invalid ENDM statement
		Cause	ENDM entered in the wrong position.
		User action	Enter ENDM in the correct position.
No.44	Code V	Message	Invalid value
		Cause	Incorrect value is described.
		User action	Enter the correct value.
No.45	Code T	Message	Invalid type
		Cause	Incorrect type expression is described.
		User action	Enter the correct type expression.
No.46	Code B	Message	Invalid BANK number
		Cause	Incorrect BANK number is described.
		User action	Enter the correct BANK number.
No.47	Code R	Message	ROM address error
		Cause	Source is too large for the ROM address.
		User action	Make the source shorter.

No. 48	Code O	Message	ORG address error
		Cause	The operand value is smaller than the immediately preceding value.
		User action	Enter a value which is larger than the preceding address value.
No. 49	Code R	Message	Used reserved word
		Cause	The reserved word has been defined as a new symbol.
		User action	Change the symbol name to something different from the reserved word.
No. 50	Code R	Message	No reserved word
		Cause	Reserved word has not been described in correct position.
		User action	Enter the correct reserved word.
No. 51	Code I	Message	Invalid data length
		Cause	Number of characters greater than permitted value
		User action	Enter the correct number of characters.
No. 52	Code N	Message	Include nesting error
		Cause	More than eight levels of include
		User action	Ensure that there are no more than eight levels of includes
No. 53	Code O	Message	Duplicated OPTION statement
		Cause	OPTION block is duplicated.
		User action	Enter one option block only per source program.
No. 54	Code M	Message	Macro area overflow
		Cause	MACRO area is insufficient.
		User action	Make the macro definition statements smaller, or, reduce the number of symbols in the macro, or, extend the area.
No. 55	Code R	Message	REPT area overflow
		Cause	Repeat area is insufficient.
		User action	Make the repeat definition smaller, or, enlarge the area.



No.56	Code I	Message	Invalid OPTION group number
		Cause	The OPTION group number is incorrect.
		User action	Enter the correct number.
No.57	Code S	Message	Symbol multi defined
		Cause	The defined symbol is in duplication.
		User action	Describe a different name for the symbol.
No.58	Code S	Message	Undefined symbol
		Cause	The symbol described has not been defined.
		User action	Enter a defined symbol, or, define the symbol.
No.59	Code P	Message	Invalid Pseudo
		Cause	Directive has not been correctly described.
		User action	Enter the directive correctly.
No.60	Code M	Message	Invalid mnemonic
			Mnemonic has not been correctly described.
		User action	Enter the correct mnemonic.
No.61	Code F	Message	include file open error
		Cause	No file, or, area insufficient.
		User action	Designate the correct include file; or, increase the memory area.
No.62	Code S	Message	parser stack overflow
		Cause	Parser stack is insufficient.
		User action	Make sure the stack level is 12 or less.
No.63	Code B	Message	Bank unmatched
		Cause	A flag has been entered with a different bank number from the operand in a built-in macro.
		User action	Provide a flag with the same bank number.
No.64	Code W	Message	No EOF statement
		Cause	No EOF in a include file.
		User action	Enter EOF at the end of the file.

No.65	Code A	Message	Statement after END
		Cause	There is a statement after an END statement.
		User action	Delete the statement after the END statement.
No.66	Code W	Message	Statement after EOF
		Cause	There is a statement after an EOP statement.
		User action	Delete the statement after the EOF statement.
No.67	Code A	Message	Address error
		Cause	Address designation is incorrect.
		User action	Designate an address which is permitted by the product.
No.68	Code W	Message	Operation in OPTION
		Cause	An instruction other than an OPTION designation has been entered in an OPTION block.
		User action	Delete the instruction.
No.69	Code C	Message	Invalid CASE LABEL
		Cause	A label other than a numeric value label has been entered in a CASE block.
		User action:	Delete the label
No.70	Code O	Message	Invalid operand
		Cause	The operand has not been described correctly.
		User action	Enter a correct operand.
No.71	Code O	Message	Illegal first operand type and value
		Cause	Type and value of the first operand are incorrect.
		User action	Enter a correct operand.
No.72	Code O	Message	Illegal second operand type and value
		Cause	Type and value of the second operand are incorrect.
		User action	Enter a correct operand.

No.73	Code O	Message	Illegal third operand type and value
		Cause	Type and value of the third operand are incorrect.
		User action	Enter a correct operand.
No.74	Code U	Message	Undefined first operand symbol
		Cause	First operand symbol has not been defined.
		User action	Enter a correct operand.
No.75	Code U	Message	Undefined second operand symbol
		Cause	Second operand symbol has not been defined.
		User action	Enter a correct operand.
No.76	Code U	Message	Undefined third operand symbol
		Cause	Third operand symbol has not been defined.
		User action	Enter a correct operand.
No.78	Code W	Message	Unsuitable for SIMPLEHOST
		Cause	BR, @ AR et. entered when using the EPA area; no guarantee that this will operate correctly in the SIMPLEHOST environment.
		User action	Refer to the appendix on the simple host.
No.79	Code 1	Message	ROM address overflow, EPA bit on
		Cause	ROM area is insufficient
		User action	Refer to the appendix on program memory overflow messages.
No.80	Code P	Message	Invalid EXIT statement
		Cause	EXIT statement entered in other than IF block. Or, two EXIT statements entered in IF block.
		User action	Delete the EXIT, or, enter a correct IF block.

No.81 Code B	Message	Boundary error
	Cause	Address boundary is incorrect.
	User action	Alter the lower 4-bits to an address other than 0FH with the DCP instruction.
No.82 Code I	Message	Illegal character
	Cause	Characters input are incorrect. (DCP instruction)
	User action	Input characters permitted by DCP.
No.83 Code F	Message	Illegal format
	Cause	Format is incorrect.
	User action	Enter a correct statement.
No.84 Code W	Message	May be shortened BR
	Cause	Branch instruction could be made shorter.
	User action	Refer to the built-in macro SKTn.
No.85 Code P	Message	Invalid ENDP statement
	Cause	ENDP has been described incorrect.
	User action	Enter correctly.
No.86 Code I	Message	Illegal use of EXTERN
	Cause	EXTERN has been used incorrect.
	User action	Enter correctly.

### List of Error Messages

<u>No.</u>	<u>ID</u>	<u>Error message</u>
11	O	Illegal first operand type
12	O	Illegal second operand type
13	O	Illegal third operand type
14	V	Illegal first operand value
15	V	Illegal second operand value
16	V	Illegal third operand value
17	S	Must be comma
18	R	Out of address range
19	A	Illegal addressing
20	W	Unreferenced symbol
21	P	No IF statement
22	P	No CASE statement
23	P	No REPT statement
24	P	No IRP statement
25	S	Symbol define error
26	A	Invalid address
27	P	No OPTION statement
28	P	No END statement
29	P	No ENDIF statement
30	P	No ENDCASE statement
31	P	No ENDR statement
32	P	No ENDM statement
33	P	No ENDP statement
34	P	No ENDOP statement
35	N	Nesting overflow
36	O	Operand count error
37	S	Syntax error
38	M	Syntax memory overflow
39	S	Symbol area overflow
40	P	Invalid EOF statement
41	P	Invalid ENDR statement
42	P	Invalid EXITR statement
43	P	Invalid ENDM statement
44	V	Invalid value

---

45	T	Invalid type
46	B	Invalid BANK number
47	R	ROM address error
48	O	ORG address error
49	R	Used reserved word
50	R	No reserved word
51	I	Invalid data length
52	N	Include nesting error
53	O	Duplicated OPTION statement
54	M	Macro area overflow
55	R	REPT area overflow
56	I	Invalid OPTION group number
57	S	Symbol multi defined
58	S	Undefined symbol
59	P	Invalid Pseudo
60	M	Invalid mnemonic
61	F	Include file open error
62	S	Parser stack overflow
63	B	Bank unmatched
64	W	No EOF statement
65	A	Statement after END
66	W	Statement after EOF
67	A	Address error
68	W	Operation in OPTION
69	C	Invalid CASE LABEL
70	O	Invalid operand
71	O	Illegal first operand type and value
72	O	Illegal second operand type and value
73	O	Illegal third operand type and value
74	U	Undefined first operand symbol
75	U	Undefined second operand symbol
76	U	Undefined third operand symbol
77	?	Unprintable error
78	W	Unsuitable for SIMPLEHOST
79	1	ROM address overflow, EPA bit on
80	P	Invalid EXIT statement
81	B	Boundary error

82	I	Illegal character
83	F	Illegal format
84	W	May be shortened BR
85	P	Invalid ENDP statement
86	I	Illegal use of EXTERN





## APPENDIX 1

## Error messages generated when program memory overflows

It sometimes occurs during program debugging that the size of the program overflows the capacity of the ROM. This is inconvenient as it does not allow the section which overflows ROM capacity to be debugged at that time.

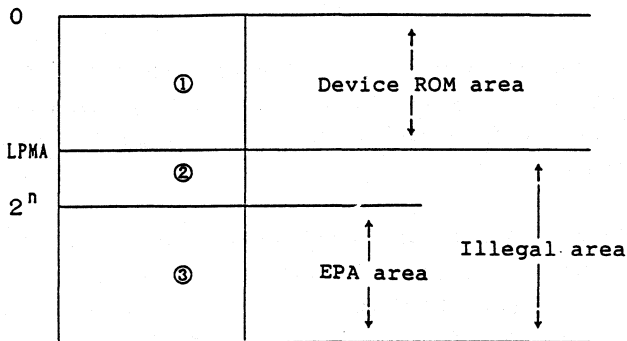
One of the uPD17000 series software development tools, the IE-17K in-circuit emulator uses a target chip program counter for debugging. However, when permitted ROM capacity is exceeded, (and the limitation with the chip is little over  $2n$ , the program counter overflows and control of the program is lost.

To counteract this problem, the IE-17K is provided with an EPA (extra program address) bit, which permits control of a program which is up to twice the volume of the target chip ROM. Thus, even though ROM capacity is exceeded, program debugging is still possible with the IE-17K.

The AS17K deals with this problem by generating object codes so that even though the program overflows ROM capacity, it is possible to address almost twice the volume of the ROM capacity. However, if there is a ROM overflow, in reality it is impossible to get the device to work, and the AS17K outputs error messages as shown below.

When generating ROM codes, program sizes should be reduced so that these errors are not output.

The output pattern for error messages generated when program memory overflows is as shown below



Note: LPMA is last program memory address

[Explanation]

In the above case, the device ROM capacity is not  $2^n$ . LPMA indicates the last address in the ROM. If the program overflows from ①, and extends into the area ③, an object code will be generated, but an error will be generated in respect of statements ② and ③.

(1) If the program is in area ②

An R error (out of address range) will be generated for statements with addresses coming after LPMA + 1.

(2) If the program is in area ③

When linking, the error message "program memory overflow, EPA bit on" will be output. In addition to this, 1 will be printed at the start of the statement line (E field) for area ③ in the assemble list. However, if there is an address referenced by a BR instruction operand in area ①, nothing will be printed in the E field. Further, even though it is in area ①, in regard to the statement referring to the address in the ③ field, 1 will be printed in the E field.

(3) If the program overflows area ③

An R error (ROM address error) will be generated in respect of each statement. An object code will not be output for statements in this field.

### APPENDIX 2 SIMPLEHOST

The uPD17000 series offers a debugger which operates under a version of MS-WINDOWS™ called SIMPLEHOST as an IE-17K host program. This permits the host to carry out break and trace operations, and memory amendments for object codes generated by the AS17K. It sends object codes to the IE-17K in real time and executes them through the SE board.

Although the SIMPLEHOST has commands which are difficult to remember, it is possible to debug programs with the listing images. If programs overflow the EPA area, the following instructions are available but their efficacy is not guaranteed.

(1) A direct designation instruction, using operand at AR

```
MOVT   DBF, @AR  
BR     @AR     etc.
```

(2) A BR instruction which uses operation expressions in the operand.

If these are patched into programs in order to use the SIMPLEHOST for debugging using listing images, the correct addresses are altered because of the effect of the patch, and it is possible that discrepancies will be generated between listed addresses and actual addresses.

The AS17K deals with this by providing a warning if one of the instructions mentioned above is in a program when it extends to the EPA area. If the warning "unsuitable for SIMPLEHOST" is generated when linking, the problem can be handled in the light of the above points.

MS-WINDOWS™ is a trademark of Microsoft Corporation.



**IE-17K**

**User's Manual**



### CHAPTER I GENERAL INFORMATION

#### 1.1 Overview

IE-17K is a software development support tool for use with all uPD17000 series 4-bit single chip micro-computers. Dedicated SE boards for each product in the series are provided. SE board, with hardware emulation function specified for each product, can also be used for program evaluation. The IE-17K consists of two boards: a memory board and a supervisor board, and it can be connected with terminals to operate as a stand alone system. In addition, by connecting to a host machine, and using Simple Soft as a man-machine interface software, a powerful debug environment can be created.

### 1.2 Characteristics of IE-17K

#### 1.2.1 Interface with target system

The use of object products for target machine interface provides the same electrical feature as of object products.

#### 1.2.2 Program Memory

The CMOS static RAM on the SE board is used for program memory.

#### 1.2.3 How TO Emulate

Two ways of program emulation can be performed in two ways: in real time emulation, 1-step emulation.

#### 1.2.4 Break function

##### (1) Programmable Break function

The following programmable Break function can be set.

- ① Break when single condition is satisfied.
- ② Setting several conditions (up to four) break selected if one or all are satisfied.
- ③ Setting several conditions (up to four) break selected if one is satisfied.
- ④ Break if conditions are satisfied in the same order as they set. The following break conditions can be set:
  - . Program memory address
  - . Data memory address
  - . Contents of data memory
  - . Address of register file
  - . Contents of resister file
  - . Command code
  - . Stack level
  - . Status of external terminal (logic analyzer)
  - . Interrupt
  - . DMA
  - . Number of executed instructions
  - . Number of conditions satisfied



(2) Error detection function

A function which issues Breaks and Warnings when a program accesses a source which is not valid for program developing the objective product. Faults detected are:

- . Invalid memory access
- . Invalid system register accessed
- . Overflow / underflow in stack level
- . Read or test memories to which no data has been written.

1.2.5 Real time trace function

A function which stores the results of execution in real time, covers a trace memory size of 32 K steps.

(1) Trace data is as follows.

- . Program memory address
- . Code of executed instruction
- . Skipped instruction
- . Written data memory address
- . Contents of written data memory
- . Status of logic analyzer terminal and relative execution time of each instruction

(2) On/off condition can be set

1.2.6 Data Memory coverage function

Memorize to which address data memory is written.

Using this function enables you the following information to be obtained.

- . Unwritten bits
- . Bits to which "1" is written
- . Bits to which "0" is written
- . Bits to which both "0" and "1" were written

### 1.2.7 Program memory coverage function

This function memorizes how many times each program address is executed. Maximum counter values is 255, even if executed more than 255 time. The counter is incremented when command of its address is executed without being skipped or referred with table reference command (MOVE, etc.). Skipped command causes the counter to increment.

### 1.2.8 Programmable pattern generator function

IE-17K contains a programmable a 14 channel pattern generator. The number of programmable steps is 8 K steps and the cycle time can be set from 1 us/step to 1333 us/step in increments of about 1 micro second.

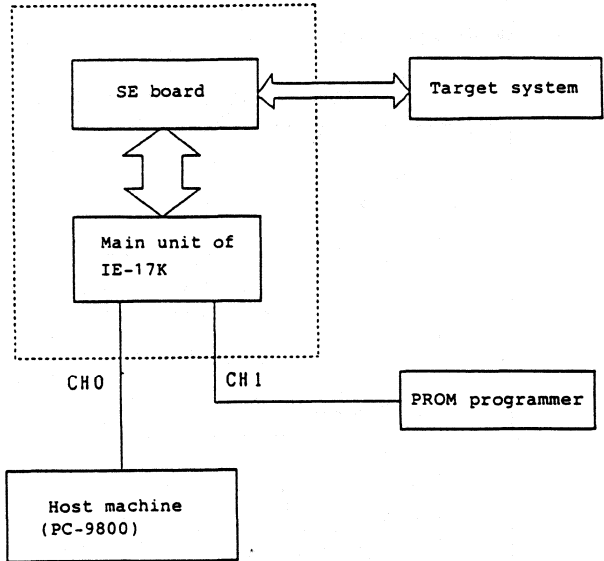
### 1.2.9 Other characteristics

- (1) The unit is provided with two RS-232C serial channels: one channel for the console and one for the PROM programmer. By connecting channel 0 to a PC-9800 and operating a Simple Host as a man-machine interface software provides a powerful debugging environment.
- (2) EMI measure was taken to satisfy VCCI standard.
- (3) Unit size 21 x 30 x 10 cm (A4 sized).
- (4) The unit contains a Switching Regulator enabling it to use commercial power.
- (5) IE-17K has enough space for installing probes.

## 1.3 Configuration

### 1.3.1 System configuration diagram

Fig. 1-1 IE-17K System Configuration Diagram



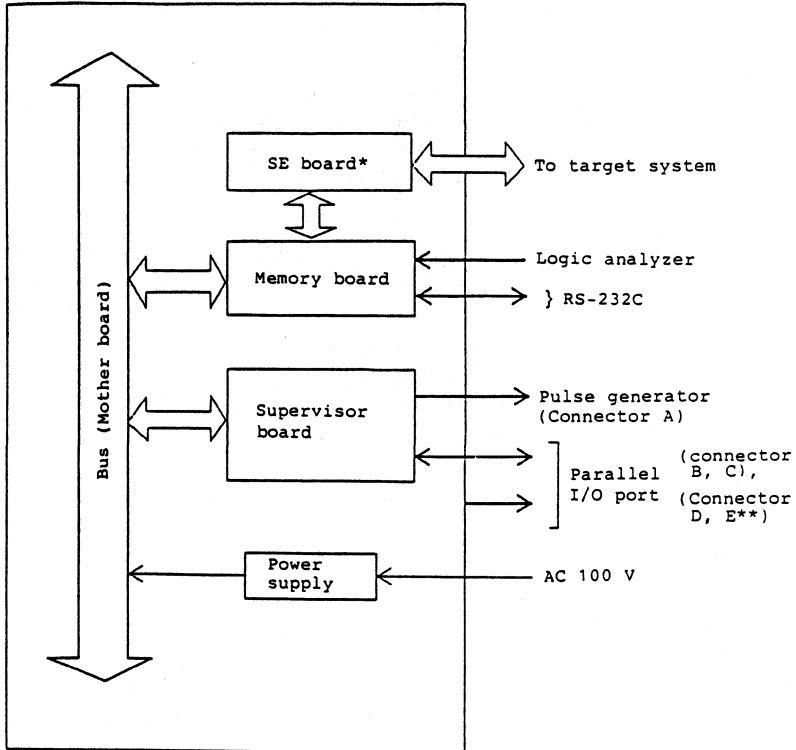
1.3.2 Block diagram

IE-17K consists of a main unit and auxiliary components.

The main unit includes the following components.

- . Frame (includes connector for connection, switches)
- . Integrated power supply
- . Supervisor (SV) board
- . Memory board
- . Mother board

Figure 1-2 IE-17K Block diagram



\* SE board is provided for each product type.

\*\* Auxiliary board

## CHAPTER 2 SPECIFICATION

### 2.1 Main LSI

<Supervisor board>

Supervisor CPU	uPD70116D	x1
Supervisor peripheral	uPD71011C	x1
Supervisor peripheral	uPD71086C	x2
Supervisor peripheral	uPD71055G	x2
Monitor ROM	uPD27C512D	x2
Monitor ROM	uPD41256V	x16
Monitor ROM	uPD4364G	x2

<Memory board>

Memory	uPD43256G	x15
Memory	uPD4364G	x1
Memory	uPD71059G	x1
Memory	uPD71054G	x2
Memory	uPD71051G	x2
Memory	uPD71082C	x2

### 2.2 Console Interface

RS-232C x 2CH (CH0, CH1)

Baud rate: 110, 300, 600, 1200, 2400, 4800, 9600, 19200 baud

Character length: 7, 8 bits

Stop bit length : 1, 2 bits

Parity : None, Even, Odd

### 2.3 Environment

Operating temperature 10 to 40°C.

Storage temperature -10 to 50°C. (Without condensation)

### 2.4 Power supply

85 to 132V AC

### 2.5 Built-in Power Supply

+5 VDC 2.0 A (Max)

+12 VDC 0.2 A (Max.)

2.6 Power consumption for Each Board

<Memory Board>

+5 VDC 110.0 mA (TYP.)

+12 VDC 32.5 mA (TYP.)

<Supervisor Board>

+5 VDC 1140.0 mA (TYP.)

2.7 External dimension (excluding projection)

Frame 210 x 300 x 100 mm







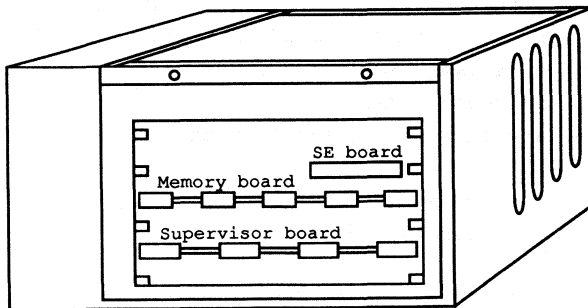
## CHAPTER 3 INSTALLATION

### 3.1 Removing Memory Board Supervisor

The body of IE-17K consists of two boards. Usually these boards are not removed. If IC exchange or switch setting requires removal of these boards, follow this procedure.

- (1) Remove the outside panel (remove the screws fixing the panel).
- (2) Remove the screws fixing the inner side board to the unit, and remove the inner board from IE-17K.
- (3) In case SE board is mounted on memory board, first, remove the SE board from the memory board.
- (4) Remove any cables connected to the memory/supervisor boards.
- (5) Remove the supervisor board via the bottom slot by pulling the card puller.
- (6) Remove memory board via the top slot by pulling the card puller.

Fig. 3-1 Mounting position of each Boards



#### <Caution of mounting/dismounting boards>

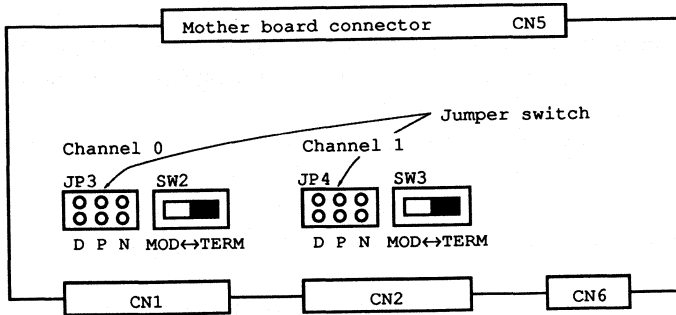
When dismantling the board, dismantle the supervisor board first, and when mounting the board, mount the memory board first.

3.2 Setting Switches

Switches on each board are set as follows.

3.2.1 Setting the switches on Memory Board

Fig. 3-2 Allocation of Memory Board Switches



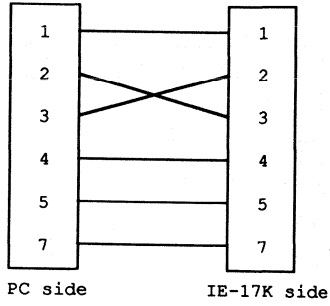
The switches on the memory board used for setting RS-232C. JP3 and SW2 are used for channel 0, and JP4 and SW3 for channel 1. JP3 and JP4 are used for changing RTS signal. Set these switches according to the host-machine used. SW2 and SW3 are for changing the terminal mode and modem mode.

These switches are factory set for shipment as follows:

- JP3, JP4 ..... Open
- SW2, SW3 ..... Terminal mode

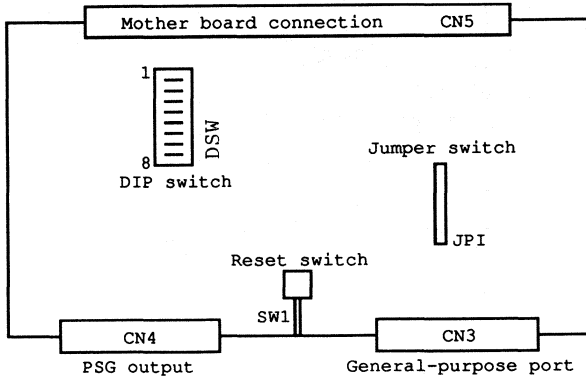
When connecting IE-17K to the PC-9800 series unit with the supplied RS-232C cable, above setting is available.

Fig. 3-4 Connection of Supplied Cables



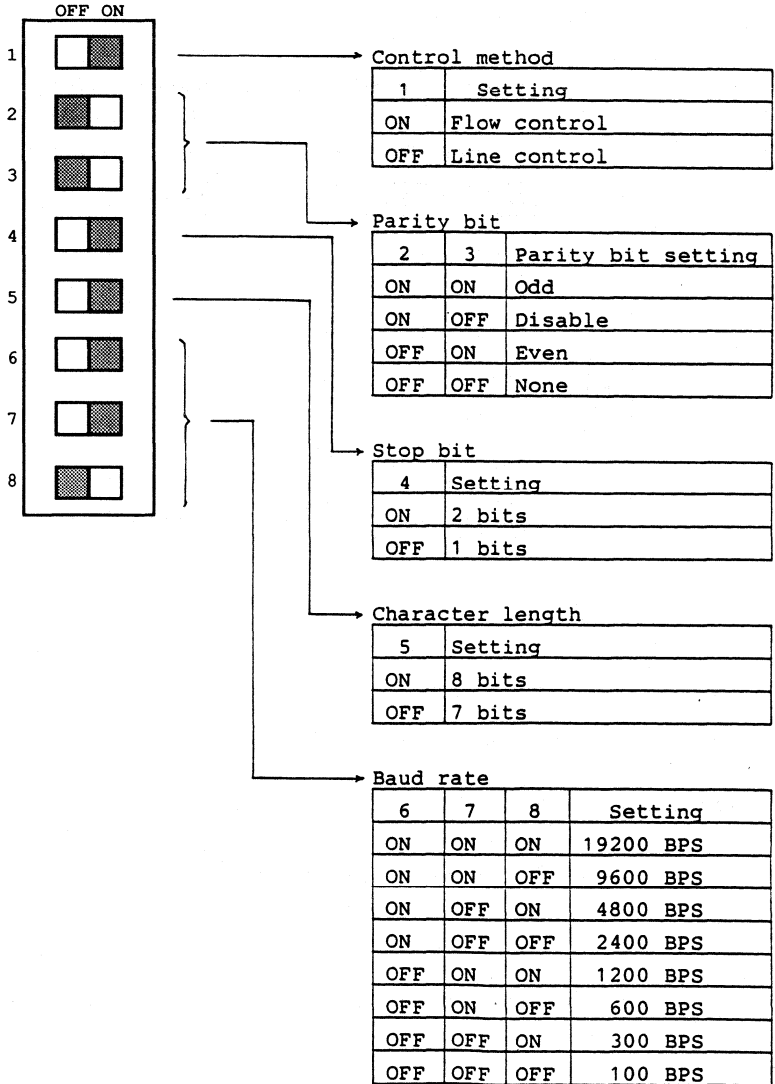
### 3.2.2 Setting switches on supervisor board


Fig. 3-5 Location of each switch on Supervisor Board



Use the DIP switch for setting RS-232C. Use JP1 as factory set for the shipping.

Fig. 3-6 Setting of DIP switch



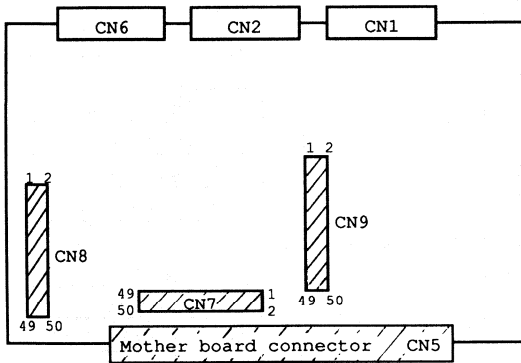
( indicates factory set switch position)

3.3 Connection of connector

3.3.1 Internal connector on memory board

Locations of connectors, CN7, CN8, CN9 on the connection section between SE Board and memory board, and connector CN5 on the mother board are shown below.

Fig. 3-7 Locations of connectors (on Memory Board)



Each connector is connected as follows:

CN1 ... Connected to the cable connector stamped "CN1".

CN2 ... Connected to the cable connector stamped "CN2".

CN6 ... Connected to 15-pin cable connector.

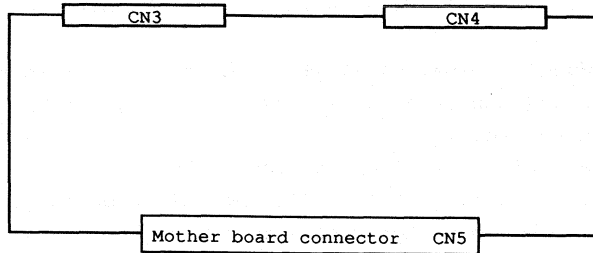
CN7 }  
 CN8 } Install on SE board.  
 CN9 }

CN5 ... Connected to the mother board.

### 3.3.2 Internal connectors on Supervisor Board

Locations of connector on supervisor board is shown as follows.

Fig. 3-8 Locations of connectors (on Supervisor Board)



Each connector should be connected as follows.

CN3 ... Connected to the 50-pin cable connector.

CN4 ... Connected to the 25-pin cable connector.

CN5 ... Connected to the mother board.

### 3.4 Installing SE board

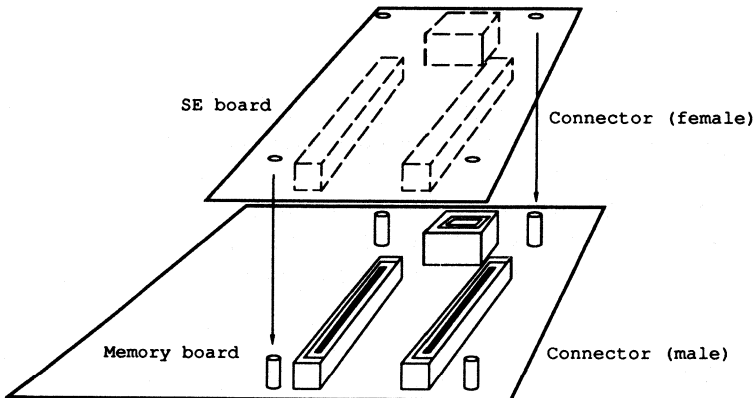
At the time of shipment, the IE-17K contains memory board and supervisor board as control boards. The SE board of which specific to each product, is not installed. Consequently it is required to install SE board corresponding to each type other than IE-17K in developing uPD17000 series.

See operation manual for details of each SE board.

The procedure for installing SE Board into IE-17K is as follows.

- (1) Remove the top cover of IE-17K by pulling out the fixed ratch on the top cover.
- (2) Remove fixing screws on top inner cover.
- (3) Remove the screws attached to the spacer on the memory board.
- (4) Connect the connector (male) on the memory board and connector (female) on the back of SE board. SE board can be installed by pressing it vertically against the memory board,

Fig. 3-9 Installation of SE board



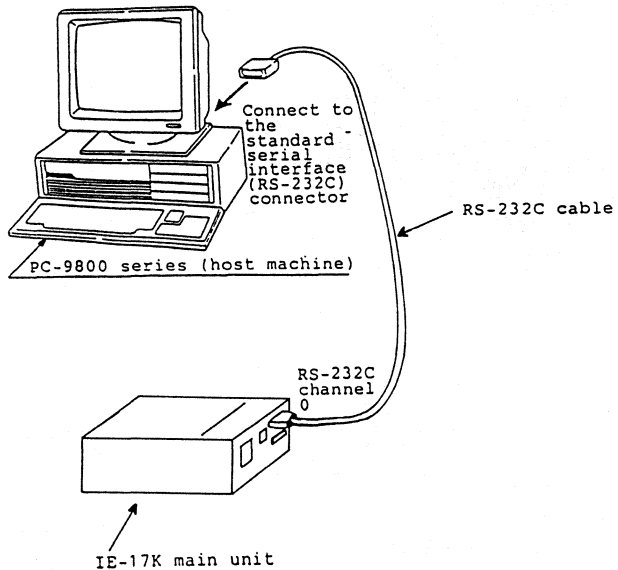
- (5) Secure SE board and memory board with the screws removed in step 2.
- (6) Replace the top inner cover and the top cover.



### 3.5 Connecting to host machine

An example of using PC-9800 series as a host machine is explained. Turn off IE-17K and PC-9800 series, connect RS-232C CHANNEL-0 connector on the IE-17K to the standard serial interface (RS-232C) connector for PC-9800 series, using the RS-232C cable supplied with the system.

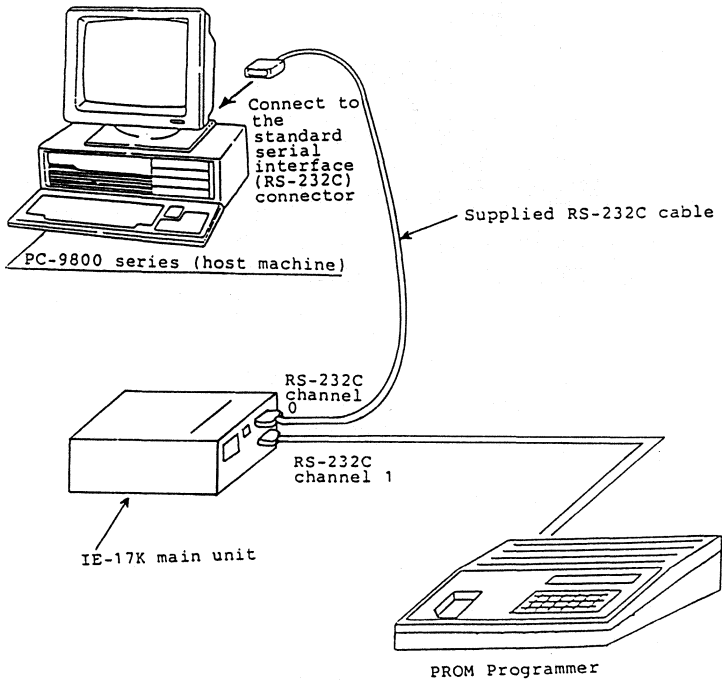
Fig 3-10 Connecting IE-17K with PC-9800 series



3.6 Connecting to PROM Programmer

Connect RS-232C CHANNEL 1 connector to PROM Programmer using RS-232C cable for PROM Programmer in order to load programs from IE-17K into PROM Programmer with IE-17K being connected to host machine (ie. PC-9800 series).

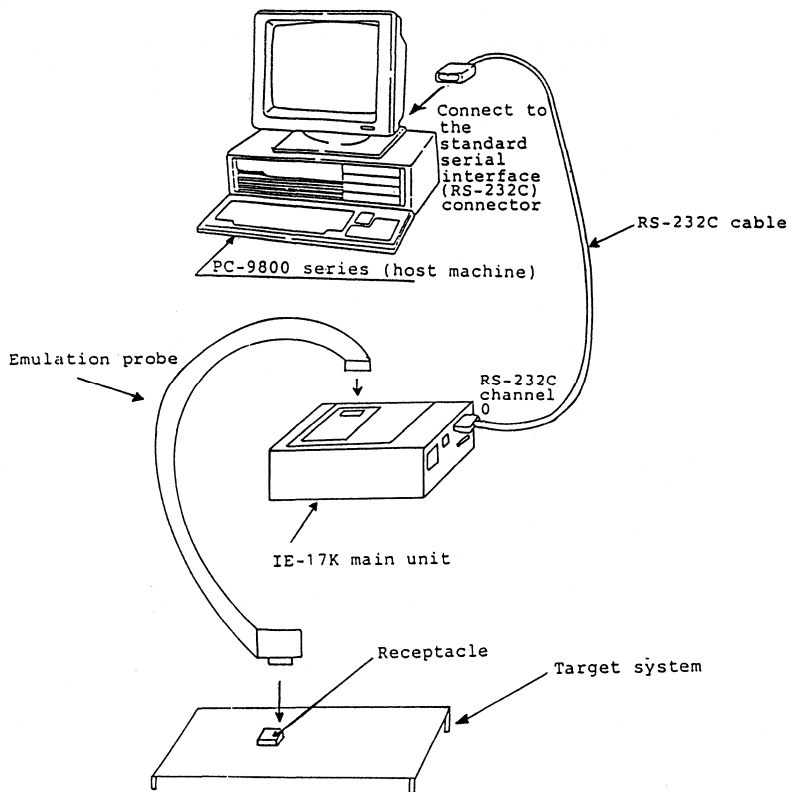
Fig 3-11 Connecting IE-17K with PROM Programmer



### 3.7 Connecting with target system

Connecting the emulation probe to the SE board, linked to a target system. For the details, see the instruction manual for each SE Board.

Fig. 3-12 Connection of IE-17K with target system





## CHAPTER 4 ACTIVATION

IE-17K is used by connecting it to a host machine or a terminal using an RS-232C cable. This chapter describes how to use the IE-17K when connected to PC-9800 series. Other host machines or terminals can be operated using the RS-232C. For details, see each device's User's manual.

Two ways of using IE-17K with PC-9800 series, are available: One is to use a terminal application such as MS-DOS WINDOW (TM) TERMINAL.EXE, and the other is to use PC-9800 series in Terminal Mode as described below. The use of terminal application enables you to load programs into IE-17K or save programs you corrected using IE-17K.

When you load save programs using a terminal application like MS-DOS WINDOW TERMINAL.EXE, refer to each terminal's User's manual.

### 4.1 Program Loading

When the IE-17K is used without a terminal program, load HEX form files created by AS-17K, using the following procedure.

- (1) Press the reset button on IE-17K.
- (2) Set the serial interface of PC-9800 series. Start MS-DOS (TM) installed in PC-9800 series, then initialize several interface by the following procedure.

```
A>>SPEED↵
SPEED Version X.X
RS232C-0 2400 BITS-7 PARITY-NONE STOP-2 NONE
-RS232C-0 9600 BITS-8 PARITY-NONE STOP-2 XON↵
A>
```

Note 1: Characters underlined to be entered via the keyboard.

Note 2: MS-DOS and MS-WINDOW are trade mark of Micro Soft inc.

(3) Load programs using COPYA command in MS-DOS

A>COPYA CON AUX↵

.LP0\$\$^Z↵

(^Z means to press Z while pressing  
the Control key.)

File has been transferred

A>COPYA file name.HEX AUX↵

File transfer completed

A>

(4) Changing DIP switches, enter into Terminal Mode, then press the Reset switch.

(5) Pressing "\$" key twice, confirms IE-17K operation and prompt appears.

\$\$

BRK>

If "\$\$" does not appeared, repeat steps (1) to (5).

For more information concerning SPEED command and COPYA command, refer to MS-DOS User's Manual.

## CHAPTER 5 COMMANDS

### 5.1 Command Notation

#### 5.1.1 Command input form

Commands are input as the following format.

```
xxx>command$$
```

\*

- \* Called as prompt, indicates the operating status of IE-17K. Prompt consists of three alphabetical characters and ">". Refer to "Prompt" in Section 5.2.

Enter a command next to prompt, and press **ESC** key or **\$** key twice. Pressing **ESC** key causes "\$" to be echoed back. Consecutive two \$\$ input after a command means the end of the command, and is called a "terminator". When a terminator is input, IE-17K executes the command.

Separated by delimiter ("\$\$"), multiple commands can be input successively.

The followings are the input form for this situation.

```
xxx>command$command$...$command$$
```

#### 5.1.2 Form of command expression

Characters, numerics, and symbols used for commands are defined as follows.

**Character:** Only alphanumeric characters can be used, and must be Uppercase characters. Lowercase characters are not accepted.

**Number :** Normally, a number is treated as hexadecimal number. When you input binary or decimal number, follow the procedure below.

**Binary constant:**

Expressed by placing "^B" before binary number.

(Example) ^B1010 (Indicates binary 1010)

**Decimal constant:**

Expressed by placing "^D" before decimal number.

(Example) ^D324 (Indicates decimal number 324)

Note: ^B or ^D means to type "B" or "D" while pressing CTRL (control) key.

**Operator :** Enables calculation among constants. Operators are as follows:

- + ... Addition
- ... Subtraction
- \* ... Multiplication
- / ... Division
- & ... Logical product (AND)
- # ... Logical sum (OR)
- ! ... Exclusive OR (XOR)
- ~ ... Negation (NOT)

Operators has no priority and are evaluated from left to right. However, if you want to specify a priority to an operators, use parenthesis.



### 5.2 Prompt

Prompt indicates the current status of the emulation tip. Prompt status are shown below.

- ① @@@> ... At start
- ② BRK> ... Break
- ③ RUN> ... Run
- ④ STP> ... Issued STOP command during RUN, and being STOPped
- ⑤ HLT> ... Issued HALT command during RUN, and being HALT
- ⑥ DMA> ... Run in DMA mode
- ⑦ DSP> ... Break after executing DS command
- ⑧ RES> ... Emulation Chip received reset signal, and being reset.

#### [Caution]

- (1) At step ①, IE-17K is not yet specified as the product type to be used, and thus the HEX file (which is a output file from AS17K Assembler) must be immediately loaded using .LP0 or .LP1. Loading of HEX file enables the system to be operated as a In-circuit-emulator.
- (2) If prompt is changed from "RUN>" to "BRK>", "STP>", or "HLT>", contents of commands which has been already input are output, and new commands are accepted.
- (3) Once prompt changes from "RUN>" to "STP>", "HLT>", "DMA>", or "RES>", it will not change back to "RUN>". In this case, when the next "\$\$" is input the prompt will change.

## 5.3 Commands

Symbols used in formats described in this section have the following meanings.

↓ : Line feed  
{ } : Select one of the contents enclosed in{ }.  
[ ] : Input can be omitted.  
\_\_\_\_ (bar under) : Means console input.

## 5.3.1 Program memory control command

- (1) Initialize Program Memory  
.IP (Initialize Program Memory)
- (2) Change Program memory  
.CP (Change Program Memory)
- (3) Dump Program memory  
.DP (Dump Program Memory)
- (4) Find Program memory  
.FP (Find Program Memory)
- (5) Save Program Memory  
.SP (Save Program Memory)
- (6) Load Program memory  
.LP (Load Program Memory)
- (7) Verification of program memory  
.VP (Verify Program Memory)
- (8) Output of PROM data  
.XS (Save PROM Data)

<code>.IP</code> Initialization of Program memory
---

Format : [  $\alpha$  ],  $\beta$ ,  $\gamma$ . | P

$\alpha$  : Start address

$\beta$  : End address

(if  $\alpha \leq \beta$ ,  $\alpha > \beta$ , error)

$\gamma$  : Data to be initialized (1-4-3-4-4 bit format).

Function : Replace the contents of address  $\alpha$  to  $\beta$  with  $\gamma$ .

If  $\alpha$  is zero,  $\alpha$  can be omitted.

Example 1: Change the contents of address 10H to 20H into 074F0.

BRK>10,20,074F0.IP\$\$

Example 2: Change the contents of address 0H to 20H into 120FF.

BRK>,20,120FF.IP\$\$

.CP Change Program Memory

Format : [  $\alpha$  ], CP

$\alpha$  : Program memory address of contents to be changed.

Function : Changes the contents of program memory address  $\alpha$ .  
If  $\alpha$  is zero,  $\alpha$  can be omitted.

Example : Change the contents of address starting from 100.

BRK>100.CP\$\$

0100:074F0-120F5 074F0-14001 074F0-11000 074F0-06100

0104:074F0-{\downarrow , \$\$}

If a value of 5-digits 14344 form is input, cursor moves to the next address. When you end input, enter "\downarrow" or "\$\$" without inputting number.

BRK>100.CP\$\$

0100:074F0-074F0 074F0-\_\_

↑

Press space key.

If space bar is used instead of number input, cursor will moved to the next address without changing the contents of the program. If wrong number is input, pressing "DEL" or "BS" key corrects the wrong input.

0100:120AF-120A1 074F0-12\_ Press "DEL" key  
↓  
0100:120AF-120A1 074F0-12\_ Press "DEL" key  
↓  
0100:120AF-120A1 074F0-1\_ Press "DEL" key  
↓  
0100:120AF-120A1 074F0-\_ Press "DEL" key  
↓  
0100:120AF-\_ Press "DEL" key  
↓  
00FF:120C1-\_-

Note: \_: Cursor

.DP      Dump Program Memory
------------------------------

Format    : [ $\alpha$ ][, $\beta$ ].DP

$\alpha$  : Start address ( $\alpha \leq \beta$ )

$\beta$  : End address ( $\alpha > \beta$ )

Function : Dump contents of program on addresses  $\alpha$  to  $\beta$ .

If address  $\alpha$  is zero, it can be omitted. If " $\beta$ " is omitted, end address become  $\alpha + \beta$ .

Example 1: Dump the contents of addresses 10H to 20H in bit format of 1-4-3-4-4.

BRK>10.20.DP\$\$

└0010:874F0 874F0 874F0 874F0 874F0 874F0 874F0 874F0

└0018:874F0 874F0 874F0 874F0 874F0 874F0 874F0 874F0

└0020:874F0 874F0 874F0 874F0 874F0 874F0 874F0 874F0

Example 2: Dump the contents of addresses 0H to 10H.

BRK> ,10.DP\$\$

0000:0C3A0 074F0 0C127 074F0 1D7E0 08042 074E0 1D704

0008:1D710 1D720 1D730 1D791 1D700 1D710 1D720 1D730

0010:1D790

Example 3: Dump the contents of addresses beginning with 10H.  
(Dump from 10H to 10H + 3FH)

BRK>10.DP\$\$

0010:1D790 1D7D0 1D7E0 074F0 074F0 074F0 074F0 167E0

0018:1D770 08770 10771 08771 10771 08772 10771 08773

0020:10771 08774 10771 08775 10771 08776 10771 08777

0028:10771 08778 10771 08779 10771 0877A 10771 0877B

0030:10771 0877C 10771 0877D 10771 0877E 10771 0877F

0038:1D000 074F0 074F0 074F0 074F0 074F0 074F0 074F0

0040:1D7F0 00000 074F0 074F0 0B7D0 097E0 0C049 1C146

0048:0C050 097F2 0C170 09000 0C171 18770 09770 0C172



.SP0, .SP1 Save Program Memory

Format : { .SP0 }  
          { .SP1 }

RS-232C Line 0: SP0

Line 1: SP1

Function : Output the contents of program memory to the RS-232C line specified by .SP0 or .SP1. Output format is same as HEX file format of AS17K.

Example : Output the contents of program memory to the line 1.

```
BRK> .SP1$$  
:1000000063A03CF061273CF0EFE040423CE0EF04AD  
:10001000EF10EF20EF30EF91EF00EF10EF20EF3017  
:10002000EF90E8C0E8D0E8E0E8F03CA138A538A6B9  
:1000300038A738E0E820E8303CF03CF080219030F0  
:10004000F7F4601C38E0B204E8E0E8F038A538E0E6  
:10005000E830E82038E08031902038E0F6F4605055
```



.LP0 .LP1 Load Program Memory

Format : { .LP0 }  
          { .LP1 }

RS-232C Line 0: LP0

RS-232C Line 1: LP1

Function : Input the contents of HEX file AS17K via RS-232C line specified by .LP0 or .LP1., or input program via Line 0.

Example : Input the program via line 0.

@@@>.LP0\$\$

Note : . At power ON or reset of IE-17K (prompt appears as "@@@>"), load HEX file AS17K with .LP command.  
      . If a program loaded by this command occupies only a part of the program memory, parts of previous program will remained in the memory.  
      . Program coverage will be cleared up.

<code>.VP0 .VP1 Verify Program Memory</code>
--

Format : { .VP0 }  
          { .VP1 }

RS-232C Line 0: VP0

Line 1: VP1

Function : Verify the contents of Program Memory and data in AS17K's HEX file from RS-232C Line specified by .VP0 or .VP1. If they are identical, "Verify OK" will be displayed, if not "Verify NG" will be displayed.

Example : Verify programs input through Line 0.

BRK> .VP0\$\$

Verify OK

Note : . If data memory information is not identical, "Verify NG DATA INITIAL VALUE" will appeared.  
      . If EPA is not identical, "Verify NG EPA" will appeared.  
      . If IFL and DFL are not identical, "Verify NG IFL DFL" will appeared.

.XS0 .XS1 Output data for PROM (Save PROM Data)

Format : { .XS0 }  
          { .XS1 }

Function : Output the contents of program memory to the RS-232C  
          Line specified by .XS0 or .XS1 with the file format of  
          AS17K PROM file.

Example : Output the contents of program memory.

```
BRK>.XS1$$  
:1000000063A03CF061273CF0EFE040423CE0EF04AD  
:10001000EF10EF20EF30EF91EF00EF10EF20EF3017  
:10002000EF90E8C0E8D0E8E0E8F03CA138A538A6B9  
:1000300038A738E0E820E8303CF03CF080219030F0  
:10004000F7F4601C38E0B204E8E0E8F038A538E0E6  
:10005000E830E82038E08031902038E0F6F4605055
```

**5.3.2 Control command for data memory**

**(1) Initialization of data memory**

**.ID (Initialize Data Memory)**

**(2) Change of data memory**

**.CD (Change Data Memory)**

**(3) Dump of data memory**

**.DD (Dump Data Memory)**

**(4) Dump of all data memory**

**.D (Dump All Data Memory)**

<code>.ID</code> Initialize Data Memory
---

Format    :  $[\alpha], \beta, \gamma, ID$   
           $\alpha$  : Start Address  
           $\beta$  : End Address  
           $\gamma$  : Contents

Function : Initialize the contents of the address from  $\alpha$  to  $\beta$  with  $\gamma$ .

Example 1: Initialize the contents of the address from 0H to 20H with 0.

BRK>10,20,0.ID\$

Example 2: Initialize the contents of the address from 0H to 20H with 1.

BRK>20,1.ID\$

.CD      Change Data Memory

Format    : [ $\alpha$ ].CD  
            $\alpha$  : Data memory address to be changed

Function : Data memory address to be changed. If  $\alpha$  is less than 0, it can be omitted.

Example  : Change the contents of address beginning from 0.

```
. BRK> .CD
0000  0-0  1-0  2-0  3-0  4-0  5-0  6-0  7-0
0008  8-0  9-0  $$↓
```

If data is input, cursor automatically moves to the next address. To end, just input "↓" or "\$\$" without numeric input.

```
. BRK> 100.CD
┌0100 ┌3-3 ┌2-
  ↑      ↑
Bank   Press space key
```

If space key is pressed instead of pressing numeric key, cursor moves to the next address without changing data contents.

If numeric input is mistaken, press "DEL" or "BS" key to correct the data.

0010:2-3 4-5 6- \_ Press "DEL" key.

↓

0010:2-3 5- \_ Press "DEL" key.

↓

0010:3- \_ Press "DEL" key.

↓

000F:4-

\_ : Cursor

<code>.DD</code> Dump Data Memory
-----------------------------------

Format    : [ $\alpha$ ] [, $\beta$ ], DD

$\alpha$  : Start address (must be  $\alpha \leq \beta$ )

$\beta$  : End address ( $\alpha > \beta$  is error)

Function : Dump data memory  $\alpha$  to  $\beta$ .

If  $\alpha$  is 0, it can be omitted.

Example 1: Dump data memory from addresses 0 - 80H.

```
BRK>0,80.DD$$
0000:0 0 1 C 6 0 0 0 0 0 0 0 0 0 0 0
0010:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0020:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0030:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0040:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0050:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0060:4 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0
0070:5 0 F 0 0 0 0 0 0 1 0 0 7 0 0 C 0
```

0080:2

Example 2: Dump the contents of data memory at address 30H.

(Data from address 30H to 7FH are dumped.)

```
BRK>30.DD$$
0030:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0040:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0050:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0060:4 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0
0070:5 0 F 0 0 0 0 0 0 1 0 0 7 0 0 C 0
```

Note     : . If " $\beta$ " is omitted, data from address  $\alpha$  to the last address where  $\alpha$  was assigned are dumped. If the address  $\alpha$  of the register file is specified, the file address  $\alpha$  to the last address of the register file are dumped.

. The contents of data memory not installed are expressed as "-"

. To dump address 0080 to 00BF means dumping the register file. If the register file is not installed, the status of the inner bus is displayed.



.D	Dump All Data Memory
----	----------------------

Format : .D

Function : Dump all data memory

Example : BRK>.D\$\$

```
0000:0 0 1 C 6 0 0 0 0 0 0 0 0 0 0 0
0010:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0020:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0030:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0040:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0050:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0060:4 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0
0070:5 0 F 0 0 0 0 0 1 0 0 7 0 0 C 0

0080:2 5 2 3 4 5 6 1 7 A B C D E 1
0090:0 0 0 3 4 0 6 0 0 9 A B C D E 0
00A0:0 1 2 3 4 5 6 1 0 9 A B C D E 0
00B0:0 0 2 3 4 7 7 F 0 0 A B C D E 2
```

)

Note : The register file is also dumped.

### 5.3.3 Emulation command

- (1) Reset  
    .R (Reset)
  
- (2) Program run  
    .RN (Run)
  
- (3) Program run (Reset condition)  
    .BG (Run Beginning Condition)
  
- (4) Break  
    .BK (Break)
  
- (5) Change start address of program  
    .CA (Change Start Address)
  
- (6) Step operation  
    .S (Step)
  
- (7) Display  
    .DS (Display)

.R	Reset
----	-------

Format : .R

Function : Reset the SE board

BRK>.R\$\$

- Note :
- . The contents of the register file or the data memory become same as that of the reset condition of the target products.
  - . The contents of data coverage will be cleared.
  - . Run start address becomes address 0H.

.RN      Program Run (RUN)
----------------------------

Format    : .RN

Function : The condition used for break trace is not changed. A program is started from the currently specified run start address.

Example  : BRK> .RN\$\$  
          RUN>

\_:Cursor

<code>.BG</code> Run Beginning Condition
--

Format    : `.BG`

Function : A program is started from the currently specified execution address and the condition used for break trace is reset.

<Contents to be reset>

- . Counter value used for Level 1 (reset value is 0)
- . Sequential stack initial value used for Level 2 (for initial value)
- . Break trace table (for initial value)
- . Specification of Trace on, One shot, or Trace off (for any Trace condition)
- . Level 1 condition

Example   : BRK> `.BG$$`

          : RUN>\_

                  :Cursor

.BK	Break
-----	-------

Format : .BK

Function : Stops program execution. When this instruction is executed, the contents of the system register and general purpose register are displayed.

This command can be accepted in the break state.

Example :

RUN> .BK\$\$

ADDR INSTRUCTION

0002 074F0 BREAK ... Instruction which processed break

0003 074F0 OVERRUN ... Most recently executed instruction

0004 0C004 NEXT ... Next instruction to be executed

PC SP AR WR BR MP IX

0004 3 0700 0 0 ... 000 ) System registers

PSW: DB CP CY Z IXE MPE JG

0 0 0 0 0 0 0

RP 0123456789ABCDEF

000 0000000000000320 ... General-purpose registers

.CA	Change program start address
-----	------------------------------

Format : [ $\alpha$ ].CA  
 $\alpha$ : Execution start address

Function : Changes the program execution start address.  
If  $\alpha$  is 0,  $\alpha$  can be omitted.

Example : BRK>100.CA\$\$





.DS	Display
-----	---------

Format : .DS

Function : Enables LCD display during break execution. This command is used to display the contents on the LCD. For some products currently under developed, contents displayed on the LCD disappear during break.

Example : BRK> .DS\$\$  
DSP>

Note : o Since emulation is in RUN state (the .BR instruction is repeatedly executed) and the contents of trace or coverage are not guaranteed after this command execution.  
o The break state can be resumed by pressing any key.

5.3.4 Break/trace condition control commands

- (1) Change break/trace condition  
CC (Change break/trace condition)
- (2) Change trace ON/OFF condition  
CT (Change trace ON/OFF condition)
- (3) Dump break/trace condition  
DC (Dump break/trace condition)
- (4) Dump trace table  
DT (Dump trace table)
- (5) Save break/trace condition  
SC (Save break/trace condition)
- (6) Load break/trace condition  
LC (Load break/trace condition)
- (7) Verify break/trace condition  
VC (Verify break/trace condition)

.CC	Change break/trace condition (Change Break/Trace Condition)
-----	---

Format : .CC

Function : Sets/changes break/trace condition.

Description:

Four break/trace conditions can be independently set, using the select unit. There are four select units. When setting any item in any of these four units, level 1 of the .CC command should be used. When setting each of these four units as break condition, level 2 of the .CC command should be used. When setting each of these four units as trace condition, the .CT command should be used.

To set the .CC command, conditions are set in the interactive mode.

The following describes each of these setting items. For items C-L, a carriage return is used to enter the default value for the item. \$ is used to exit from the setting mode.

<Break condition set item (for level 1)>

A) LEVEL (1, 2): ? 1

... Selects level. Two levels , level 1 and level 2 are available.

B) UNIT (0 - 3): ?

... Selects unit. Units are 0 to 3.

For items that can be set for each unit, refer to Table 5-1.

CATG (C - L): ?

... Selects condition from set items C) to L).

Some units do not have this selection. If an item which is selected, is not available the next available item will be selected.

C) CONDITION AND(1)/OR(0): Default?

- ... Selects whether sets items D) to K) will ANDed or ORed. When AND is selected, the break condition will be established when all set items in a given unit are satisfied. Therefore, if it is necessary to eliminate any condition from ANDed conditions, the condition should be set in a manner it is always satisfied. For E) and I), the timing signal is concerned with establishing the condition, if it is necessary to remove any item, specify 1 for RELEASE -- FROM AND ---.

D) PROG ADDR UPPER: Default?

- ... Specifies the top address of the break/trace range of the program.

PROG ADDR LOWER: Default?

- ... Specifies the bottom address of the break/trace range of the program.

MATCH(1)/UNMATCH(0): Default?

- ... When MATCH is specified, the program address range specified in the above will become a break/trace condition.
- ... When UNMATCH is specified, the addresses outside the program address range specified in the above will become a break/trace condition.

E) RELEASE DATAMEMORY FROM AND YES(1)/NO(0): Default?

- ... When removing item E) which is related to the data memory from ANDed condition of D) to K) (when 1 is selected in (C)), input 1. If ORed condition of D) to K) (when 0 is selected in (C)), the content of this setting is ignored so that it can be left either as 1 or 0. For conditions relating to the data memory, these three conditions are ANDed; DATA ADDR, CURRENT DATA, and PREVIOUS DATA (excluded on some units).

## DATA ADDR: Default?

... Specifies a break/trace condition with the data memory address so that a break/trace occurs when data is written to the specified address.

## DATA ADDR MASK: Default?

... Specifies mask data for the data memory address specified as a break/trace condition. The mask data is hexadecimal where 1 is set for the data memory address to be used as the break/trace condition and 0 is set to the bit which can be either 1 or 0.

Since this item is not provided for on unit 2, a data memory break/trace condition for unit 2 cannot be invalidated.

## MATCH(1)/UNMATCH(0): Default?

... When MATCH is specified, the DATA ADDR value will become a break/trace condition.

... When UNMATCH is specified, values other than the DATA ADDR value will become a break/trace condition.

## CURRENT DATA: Default?

... Specifies the break/trace condition with the value written to the data memory.

## CURRENT MASK: Default?

... Specifies the mask data for the data memory value used as the break/trace condition.

Since this item is not provided for unit on 2, a data memory break/trace condition for unit 2 cannot be invalidated.

## MATCH(1)/UNMATCH(0): Default?

... When MATCH is specified, the CURRENT DATA value specified in the above will become a break/trace condition.

... When UNMATCH is specified, values other than the CURRENT DATA value specified in the above will become a break/trace condition.

PREVIOUS DATA DISABLE YES(1)/NO(0): Default?

... DATA ADDR, CURRENT DATA, and PREVIOUS DATA conditions are ANDed as a break/trace condition. Therefore, to exclude PREVIOUS DATA condition from item E), specify 1.

PREVIOUS DATA: Default?

... Specifies a break/trace condition with the value of the data memory before data is written to memory.

MATCH(1)/UNMATCH(0): Default?

... When MATCH is specified, the PREVIOUS DATA value described in the above will become a break/trace condition.

... When UNMATCH is specified, values other than the PREVIOUS DATA value described in the above will become a break/trace condition.

F) SP LEVEL UPPER: Default?

... Specifies the top of the break/trace range in the stack pointer.

SP LEVEL LOWER: Default?

... Specifies the bottom of the break/trace range in the stack pointer.

MATCH(1)/UNMATCH(0): Default?

... When MATCH is specified, the stack pointer range described above will become a break/trace condition.

... When UNMATCH is specified, outside the stack pointer range described above will become a break/trace condition.

G) INST CODE: Default?

... Specifies a break/trace condition with the instruction code to be executed. The instruction code format is a 1-4-3-4-4-bit format.

INST MASK: Default?

... Specifies the mask data for the instruction code which will be used as a break/trace condition.

### MATCH(1)/UNMATCH(0): Default?

- ... When MATCH is specified, the instruction code specified above will become a break/trace condition.
- ... When UNMATCH is specified, codes other than the instruction code specified above will become a break/trace condition.

### H) PORT DATA: Default?

- ... Specifies a break/trace condition with the value input from the logic analyzer probe connected to connector A. The unit's pins and the logic analyzer's probe pins are as follows:

UNIT	Logic analyzer probe pin
0	→ ST0
1	→ ST1
2	→ ST2
3	→ ST3

### PORT MASK: Default?

- ... Specifies the mask data for the port data which will be used as a break/trace condition.

### EDGE(1)/LEVEL(0): Default?

- ... When 0 is specified for PORT DATA, the falling edge becomes a break/trace condition if EDGE is specified. If LEVEL is specified, a low level (pin level) becomes a break/trace condition.

When 1 is specified for PORT DATA, the rising edge becomes a break/trace condition if EDGE is specified. If LEVEL is specified, a high level (pin level) becomes a break/trace condition.

### MATCH(1)/UNMATCH(0): Default?

- ... When MATCH is specified, the PORT DATA status specified in the above will become a break/trace condition.
- ... When UNMATCH is specified, status other than the PORT DATA status specified above will become a break/trace condition.

XREQ DATA: Default?

... Specifies a break/trace condition with the value input from the XREQ pin of the logic analyzer probe.

The XREQ pin of the logic analyzer probe is exclusively used for inputting the external break signal.

XREQ MASK: Default?

... Specifies the mask data for the XREQ data which will be used as a break/trace condition.

EDGE(1)/LEVEL(0): Default?

... When 0 is specified for XREQ DATA, the falling edge becomes a break/trace condition if EDGE is specified. If LEVEL is specified, a low level (pin level) becomes a break/trace condition.

When 1 is specified for XREQ DATA, the rising edge becomes a break/trace condition if EDGE is specified. If LEVEL is specified, a high level (pin level) becomes a break/trace condition.

MATCH(1)/UNMATCH(0): Default?

... When MATCH is specified, the XREQ DATA status specified in the above will become a break/trace condition.

... When UNMATCH is specified, status other than the XREQ DATA status specified above will become a break/trace condition.

For this item, the break/trace condition with the PORT DATA and the break/trace condition with the XREQ data are Ored.

I) This item is not currently supported. Therefore, the break/trace condition for this item must be invalidated in the following manner.

RELEASE MAR FROM AND YES(1) / NO(0) : 0 ? 1

MAR DATA : 0 ? 0

MAR MASK : 0 ? 1

MATCH(1) / UNMATCH(0) : 0 ? 0



### J) INTERRUPT ACKNOWLEDGE: Default?

... Specifies a break/trace condition with an interrupt generation. When 1 is specified, the break/trace condition will be satisfied when an interrupt is generated during program execution.

The break/trace start address for interrupt generation is the corresponding vector address.

### INTERRUPT MASK: Default?

... Specifies the mask data for the value specified for the interrupt used as a break/trace condition.

### MATCH(1)/UNMATCH(0): Default?

... When MATCH is specified, the value specified for the above interrupt will become a break/trace condition.

... When UNMATCH is specified, values other than the value specified for the above interrupt will become a break/trace condition.

### K) DMA: Default?

... Specifies a break/trace condition with a generation of DMA (Direct Memory Access).

Specify 1 to satisfy the break/trace condition when DMA is generated.

Specify 0 to satisfy the break/trace condition when DMA is not generated.

It must be noted that no break will be generated when DMA is performed.

### DMA MASK: Default?

... Specifies the mask data for the value specified for the DMA used as a break/trace condition.

### MATCH(1)/UNMATCH(0): Default?

... When MATCH is specified, the value specified for the above DMA will become a break condition.

... When UNMATCH is specified, values other than the value specified for the above DMA will become a break condition.

L) COUNTER SOURCE SELECTNO(0)/INST(1)/CONDITION(2)/INST AFTER CONDITION(3): 0?

... Specifies a break/trace condition with counter overflow.

The counter is an up-counter which increments (+1) its contents from initial value 0.

The counter can be set in the following four ways:

- . NO(0) ..... The counter will not be used.
- . INST(1) ..... Unconditionally counts the number of instruction executions.
- . CONDITION(2) .. Counts the number of executions of instructions which satisfy the break condition of the unit specified in C-K.
- . INST AFTER CONDITION(3)

... Counts the number of instructions executed after the conditions of items C) to K) are satisfied.

TERMINAL COUNTER: Default?

... Specifies the counter end value.

COUNTER MASK: Default?

... Specifies the mask data value for the value specified for a counter used as a break/trace condition.

MATCH(1)/UNMATCH(0): Default?

... When MATCH is specified, the counter value specified above will become a break condition.

... When UNMATCH is specified, values other than the counter value specified above will become a break condition.

(Output example of each unit)  
 <Unit 0>

```

BRK > .CCSS
A) LEVEL (1, 2)           : ? 1
B) UNIT (0 - 3)          : ? 0
   CATG (C - L)          : ? C
C) CONDITION AND(1) / OR(0) : 0 ?
D) PROG ADDR UPER        : FFFF ?
   PROG ADDR LOWER       : 0000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : 0 ?
   DATA ADDR            : 000 ?
   DATA ADDR MASK       : 000 ?
   CURRENT DATA         : 0 ?
   CURRENT MASK          : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
F) SP LEVEL UPER        : 0 ?
   SP LEVEL LOWER       : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
H) PORT DATA           : 0 ?
   PORT MASK            : 0 ?
   EDGE(1) / LEVEL(0)   : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   XREQ DATA           : 0 ?
   XREQ MASK            : 0 ?
   EDGE(1) / LEVEL(0)   : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
I) INTERRUPT ACKNOWLEDGE : 0 ?
   INTERRUPT MASK       : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
K) DMA                  : 0 ?
   DMA MASK             : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
L) COUNTER SOURCE SELECT
   NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3) : 0 ?
   TERMINAL COUNTER      : 0000 ?
   COUNTER MASK          : 0000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
  
```

} Program memory

} Data memory

} Stack pointer

} Logic analyzer probe ST0 pin

} Logic analyzer probe XREQ pin

} Interrupt

} DMA

} Counter

## UM IE-17K

<Unit 1>

.CC\$\$

```

A) LEVEL (1, 2)           : ? 1
B) UNIT (0 - 3)          : ? 1
   CATG (C - L)           : ? G
C) CONDITION AND(1) / OR(0) : 0 ?
D) PROG ADDR UPER        : FFFF ?
   PROG ADDR LOWER       : 0000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
} Program memory
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : 0 ?
   DATA ADDR            : 000 ?
   DATA ADDR MASK       : 000 ?
   CURRENT DATA         : 0 ?
   CURRENT MASK          : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   PREVIOUS DATA DISABLE YES(1) / NO(0) : 0 ?
   PREVIOUS DATA       : 0 ?
   MATCH(1) / UNMATCH(1) : 0 ?
} Data memory
H) PORT DATA            : 0 ?
   PORT MASK             : 0 ?
   EDGE(1) / LEVEL(0)    : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
} Logic analyzer probe ST1 pin
I) RELEASE MAR FROM AND YES(1) / NO(0) : ?
   MAR DATA             : 0 ?
   MAR MASK              : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
} MAR*
L) COUNTER SOURCE SELECT
   NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3) : 0 ?
   TERMINAL COUNTER      : 00 ?
   COUNTER MASK          : 00 ?
   MATCH(1) / UNMATCH(0) : 0 ?
} Counter

```

\* Not currently supported.

<Unit 2>

Level 1, Unit 2

BRK > .CCSS

A) LEVEL (1, 2)	: ? 1	
B) UNIT (0 - 3)	: ? 2	
CATG (C - L)	: ? C	
C) CONDITION AND(1) / OR(0)	: 0 ?	
D) PROG ADDR UPER	: FFFF ?	} Program memory
PROG ADDR LOWER	: 0000 ?	
MATCH(1) / UNMATCH(0)	: 0 ?	
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0)	: 0 ?	} Data memory
DATA ADDR	: 000 ?	
MATCH(1) / UNMATCH(1)	: 0 ?	
CURRENT DATA	: 0 ?	
MATCH(1) / UNMATCH(0)	: 0 ?	
H) PORT DATA	: 0 ?	} Logic analyzer probe ST2 pin
PORT MASK	: 0 ?	
EDGE(1) / LEVEL(0)	: 0 ?	
MATCH(1) / UNMATCH(0)	: 0 ?	
L) COUNTER SOURCE SELECT		} Counter
NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3)	: 0 ?	
TERMINAL COUNTER	: 00 ?	
COUNTER MASK	: 00 ?	
MATCH(1) / UNMATCH(0)	: 0 ?	

## UM IE-17K

<Unit 3>

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 3
   CATG (C - L) : ? C
C) CONDITION AND(1) / OR(0) : 0 ?
D) PROG ADDR UPER      : FFFF ?
   PROG ADDR LOWER    : 0000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
G) INST CODE          : 00000 ?
   INST MASK          : 00000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
H) PORT DATA         : 0 ?
   PORT MASK          : 0 ?
   EDGE(1) / LEVEL(0) : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
```

} Program memory

} Instruction code

} Logic analyzer probe ST3 pin

<Break condition set item (level 2)>

When setting each unit (units 0 to 3) specified in level 1 as the break condition, level 2 is used.

This setting is made in four hierarchical levels using a concept called DEPTH.

OR condition for four units can be set in one DEPTH. A unit for which 1 is specified will be included in the OR condition. A unit for which 0 is specified will be excluded from the OR condition. When the OR condition in one DEPTH is satisfied, the satisfaction of the next DEPTH will be awaited.

A break occurs when the condition for DEPTH0 is satisfied. Satisfaction of condition is awaited in this order; DEPTH3 -> DEPTH1.

The DEPTH from which satisfaction of condition will be awaited can be set by INITIAL DEPTH setting.

Example:

BRK>.CC\$\$

A) LEVEL.(1 , 2) : ? 2

B) LEVEL 2 : 0123

DEPTH-3 : 1101 ? 0000

DEPTH-2 : 1101 ? 1111

DEPTH-1 : 1101 ? 1010

DEPTH-0 : 1101 ? 0001

INITIAL DEPTH : 0 ? 1

This setting generates a break when the condition of unit 3 specified in level 1 is satisfied after the condition of unit 0 or 2 specified in level 1 is satisfied.

Table 5-1 List of Break/Trace Conditions

ITEM	UNIT0	UNIT1	UNIT2	UNIT3
C) CONDITION AND(1) / OR(0)	o	o	o	o
D) PROG ADDR UPER PROG ADDR LOWER MATCH(1) / UNMATCH(0)	o	o	o	o
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) DATA ADDR			o	
----- DATA ADDR MASK			-----	
----- MATCH(1) / UNMATCH(0) CURRENT DATA	o	o	o	
----- CURRENT MASK			-----	
----- MATCH(1) / UNMATCH(0)			o	
----- PREVIOUS DATA DISABLE YES(1) / NO(0) PREVIOUS DATA MATCH(1) / UNMATCH(0)		o		
F) SP LEVEL UPER SP LEVEL LOWER MATCH(1) / UNMATCH(0)	o			
G) INST CODE INST MASK MATCH(1) / UNMATCH(0)				o
H) PORT DATA PORT MASK EDGE(1) / LEVEL(0) MATCH(1) / UNMATCH(0)	o ST0	o ST1	o ST2	o ST3
----- XREQ DATA XREQ MASK EDGE(1) / LEVEL(0) MATCH(1) / UNMATCH(0)	o			
J) INTERRUPT ACKNOWLEDGE INTERRUPT MASK MATCH(1) / UNMATCH(0)	o			
K) DMA DMA MASK MATCH(1) / UNMATCH(0)	o			
L) COUNTER SOURCE SELECT NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3) TERMINAL COUNTER COUNTER MASK MATCH(1) / UNMATCH(0)	o	o	o	



.CT	Change trace ON/OFF condition
-----	-------------------------------

Format : .CT

Function : Changes the trace ON/OFF condition.

Description:

Sets each unit specified in level 1 of .CC as the trace condition. The trace ON/OFF condition is set in the following manner.

```
BRK>.CT$$  
      TRACK CONDITION  MODE  
D: TRACE DON'T CARE ..... (1)  
T: TRACE ON           ..... (2)  
U: TRACE OFF          ..... (3)  
S: TRACE ONE SHOT    ..... (4)  
LEVEL 1 UNIT : 0123  
              DDDD ?  
                    
              Default
```

- (1) Satisfaction of the break condition of the unit has no effect on trace.
- (2) Trace starts (ON) when the break condition of the unit is satisfied.
- (3) Trace ends (OFF) when the break condition of the unit is satisfied.
- (4) Trace will be performed only in the portion for which the break condition of the unit is satisfied (trace one shot).

Note : If the trace conditions of two or more units are satisfied at the same location, the following priority order is used to validate the trace condition.

TRACE ON > TRACE ONE SHOT > TRACE OFF

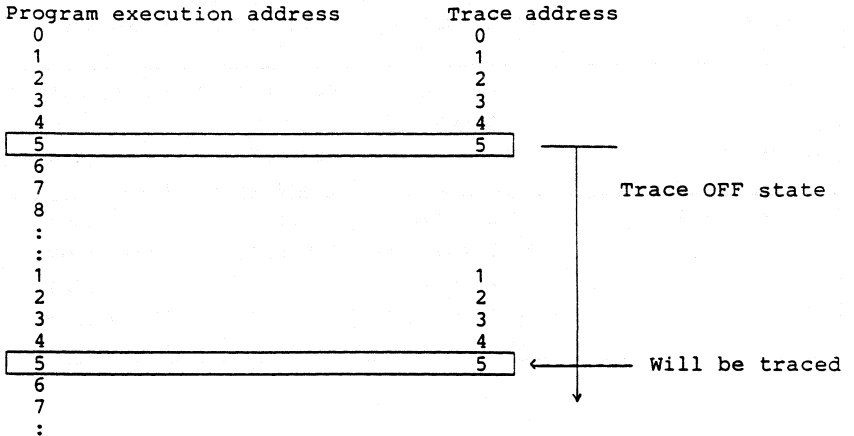
Example :

```
BRK>.CT$$  
    TRACK CONDITION  MODE  
D: TRACE DON'T CARE  
T: TRACE ON  
U: TRACE OFF  
S: TRACE ONE SHOT  
LEVEL 1 UNIT : 0123  
           : DDDD ? TUSS
```

Trace starts when the condition for unit 0 is satisfied.  
Trace ends when the condition for unit 1 is satisfied.  
Trace is performed while the condition for unit 2 or 3 is satisfied.

- Note: . There are two types of trace, address trace and status trace. Address trace is performed regardless of the setting of this command.
- . Trace ON is assumed when the execution is made after inputting .R or the execution is started with .BG.
  - . The contents set by .CT is not affected when the execution is made after inputting .R or the execution is started with .BG.
  - . After trace ON or trace OFF, trace ON or trace OFF will be maintained even if TRACE DON'T CARE is set.  
TRACE ONE SHOT is effective only in the trace OFF state.  
If TRACE ONE SHOT is specified in the trace OFF state, trace will be performed only for the address for which the condition is satisfied. Additionally, setting TRACE DON'T CARE after specifying TRACE ONE SHOT will not maintain the TRACE ONE SHOT specification, but will maintains the trace OFF status existing before specifying TRACE ONE SHOT.
  - . When trace OFF is specified, afterwards, no trace will be executed; however, the execution address which determines trace OFF will be executed (this is similar to when TRACE ONE SHOT is specified).

Example 1: When trace OFF state continues after trace OFF has been initiated at address 5H for unit 0.



Example 2: When, after trace OFF state is initiated at address 5H for unit 0, TRACE DON'T CARE is specified at the same address (5H).

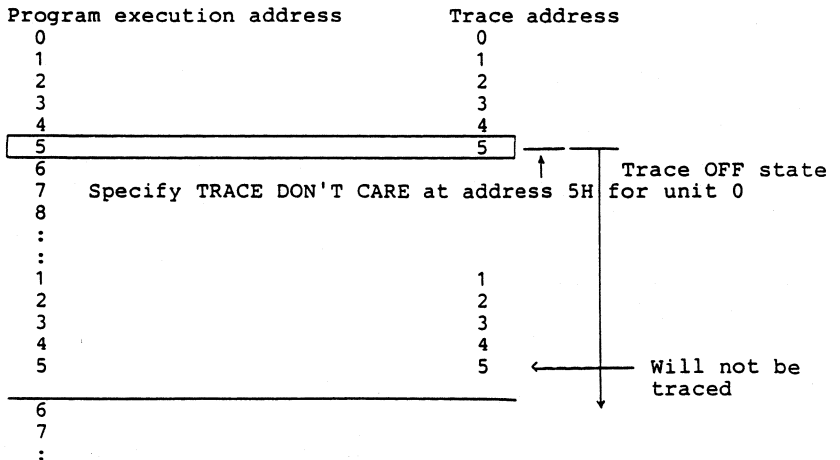


Table 5-2 Transition of Trace State

Satisfied condition \ Current trace state	During trace ON	During trace OFF	During trace ONE SHOT
Trace ON	Continues trace	Starts trace	Starts trace
Trace OFF	Ends trace	Maintains trace OFF	Ends trace ONE SHOT
Trace ONE SHOT	Continues trace (ONE SHOT is invalid)	Starts trace ONE SHOT	Continues trace ONE SHOT (most recent condition is valid)

.DC                                  Dump break / trace consitions (Dump Break Condition)

Format    : .DC

Function  : Dumps the brak / trace consitions.

Example   : To dump the break / trace conditions of untis 0 to 3:

```
BRK > .DC$$
UNIT (0 - 3) ? 0
CONDITION : OR
PROG ADDR : FFFF - 0000 UNMATCH
DATA ADDR : 000 <000> UNMATCH
CRNT      : 0 <0> UNMATCH
SP LEVEL  : F - 0 UNMATCH
PORT DATA : 0 <0> LEVEL UNMATCH   XREQ : 0 <1> LEVEL UNMATCH
INTERRUPT : 0 <0> UNMATCH
DMA       : 0 <0> UNMATCH
COUNT SEL : NO 0000 <0000> UNMATCH
TRACE SEL  : TRACE ON
```

```
BRK > .DC$$
UNIT (0 - 3) ? 1
CONDITION : OR
PROG ADDR : FFFF - 0000 UNMATCH
DATA ADDR : 000 <000> UNMATCH
CRNT      : 0 <0> UNMATCH
PRVS     : 0 UNMATCH
PORT DATA : 0 <0> LEVEL UNMATCH
MAR DATA : 0 <0> UNMATCH
COUNT SEL : NO 00 <00> UNMATCH
TRACE SEL  : TRACE OFF
```

```
BRK > .DC$$
UNIT (0 - 3) ? 2
CONDITION : OR
PROG ADDR : FFFF - 0000 UNMATCH
DATA ADDR : 000 <000> UNMATCH
CRNT      : 0 <0> UNMATCH
PORT DATA : 0 <0> LEVEL UNMATCH
COUNT SEL : NO 00 <00> UNMATCH
TRACE SEL  : TRACE DON'T CARE
```

```
BRK > .DC$$
UNIT (0 - 3) ? 3
CONDITION : OR
PROG ADDR : FFFF - 0000 UNMATCH
INST CODE : 0000 <0000> UNMATCH
PORT DATA : 0 <0> LEVEL UNMATCH
TRACE SEL  : TRACE DON'T CARE
```

Note: Contents of < > are mask data.

.DT	Dump trace table
-----	------------------

Format : [ $\alpha$ , $\beta$ ].DT

$\alpha$ : Dump start trace number       $\alpha < \beta$   
 $\beta$ : Dump end trace number           $\alpha > \beta$  will result in error

Function : Dumps the trace contents from the specified trace start number  $\alpha$  to the specified trace end number.

**Description:**

If both  $\alpha$  and  $\beta$  are not specified, the last contents of the trace table will be displayed. When the system resets or the .R command is input, the trace table will be initialized and the trace counter will be initialized to 0.

For a trace with fewer than 32K steps (32768 in decimal), the trace counter indicates the end of the trace table.

For a trace which exceeds 32K steps, the most recent 32k steps of the trace contents will be stored in the trace table, and the trace counter will indicate 7FFFH (32767).

There are two types of trace, the address trace and the status trace.

For the address trace, the most recent program execution contents will be traced regardless of the trace condition.

For the status trace, the range specified by the trace condition will be traced (the status trace will contain more informations than the address trace).

Example 1: To dump the result of the address trace from trace number 0 to trace number 10:

```
BRK>0.10.DT$$
ADDRESS (1) / STATUS (0) TRACE ? 1↓
```

TR_NO	ADDR	INSTRUCTION
00000	0000	0000 074F0
00001	0001	0001 074F0
00002	0002	0002 074F0
00003	0003	0003 074F0
00004	0004	0004 074F0
00005	0005	0005 074F0

```

00006 0006 0006 074F0
00007 0007 0007 074F0
00008 0008 0008 074F0
00009 0009 0009 074F0
00010 000A 000A 074F0
(1) (2) (3) (4)

```

Example 2: To dump the result of the status trace from trace number 0 to trace number 10:

```

BRK>0.10.DT$$
ADDRESS (1) / STATUS (0) TRACE ? 0 ↓

```

TR_NO	ADDR	INSTRUCTION	PORT	WA	DB	JG	TIME
00000	0000	0000 074F0	11111111	04F	0	0	0130001
00001	0001	0001 074F0	11111111	04F	0	0	0130002
00002	0002	0002 074F0	11111111	04F	0	0	0130003
00003	0003	0003 074F0	11111111	04F	0	0	0130004
00004	0004	0004 074F0	11111111	04F	0	0	0130005
00005	0005	0005 074F0	11111111	04F	0	0	0130006
00006	0006	0006 074F0	11111111	04F	0	0	0130007
00007	0007	0007 074F0	11111111	04F	0	0	0130008
00008	0008	0008 074F0	11111111	04F	0	0	0130009
00009	0009	0009 074F0	11111111	04F	0	0	013000A
00010	000A	000A 074F0	11111111	04F	0	0	013000B
00011	000B	000B 074F0	11111111	04F	0	0	013000C
00012	000C	000C 074F0	11111111	04F	0	0	013000D
00013	000D	000D 074F0	11111111	04F	0	0	013000E
00014	000E	000E 074F0	11111111	04F	0	0	013000F
00015	000F	000F 074F0	11111111	04F	0	0	0130010
00016	0010	0010 074F0	11111111	04F	0	0	0130011

(1) (2) (3) (4) (5) (6) (7) (8) (9)

- (1) Trace number displayed in decimal
- (2) Trace number displayed in hexadecimal
- (3) Program address (program counter value)
- (4) Instruction code (1-4-3-4-4-bit format)
- (5) Status of each pin of the logic analyzer probe.  
From right, each digit represents ST0, ST1, ..., ST7.
- (6) Data memory write address  
Effective only when data is written to the data memory.
- (7) Data bus  
Indicates the value of data written to the data memory.
- (8) Instruction skipped when the skip instruction is executed is indicated by an asterisk (\*).

(9) Time stamp

This is set to 1 when the .RN command is executed, and incremented (+1) each time an instruction is executed. However, when the MOV instruction is executed, this value will be incremented by 2 (+2).



```
.SC0 .SC1 Save break/trace condition
```

Format : { .SC0 }  
          { .SC1 }

RS-232C Channel 0: VP0  
                  Channel 1: VP1

Function : Outputs the break/trace condition specified by level 1  
          of .CC to the RS-232C channel specified by .SC0 or .SC1  
          in the Intel-HEX format.

Example : To output the break/trace condition to channel 0:

```
BRK>.SC0$$  
:104143000B0B0B0B00FF02020200000000001003A  
:1041530000010100000000FFFF0000100000FF0746  
:104163000B000F0400000F00040000FFFF0000011C  
:104173000001000F0000010100010100000000FF29  
:10418300FF000010000000FFFF0000100000FF0709  
:104193000B000F0400040F00000000FFFF000001EC  
:1041A3000001000F0400010000010000000000FFF7  
:1041B300FF000008000000FFFF0000100000FF07E1  
:1041C3000B000F0400000F00000000FFFF000001C0  
:1041D3000001000F0000010000010000000000FFCB  
:1041E300FF000008000000FFFF0000100000FF07B1  
:1041F3000000F0000000F0000000FFFF1000018F  
:104203000001000F0000010000010000000000FF9A  
:07421300FF00000000000000A5  
:00000001FF
```

.LC0 .LC1 Load break/trace condition

Format : { .LC0 }  
          { .LC1 }

RS-232C Channel 0: LC0  
          Channel 1: LC1

Function : Inputs the break/trace condition output by .SC0 or .SC1  
          from the RS-232C channel specified by .LC0 or .LC1.

Example : To input the break/trace condition from channel 0:  
          BRK> .LC0\$\$

.VC0 .VC1 Verify break/trace condition

Format : { .VC0 }  
          { .VC1 }

RS-232C Channel 0: VC0  
          Channel 1: VC1

Function : Verifies the break/trace condition against the data sent from the RS-232C line specified by .VC0 or .VC1. If they coincide, "Verify OK" will be displayed. If they do not coincide, "Verify NG" will be displayed.

Example : To verify the break/trace condition input from RS-232C channel 0:

BRK> .VCO\$\$  
Verify OK

**5.3.5 Coverage display command****(1) Dump coverage memory****DM (Dump coverage memory)**

.DM Dump coverage memory

Format : [ $\alpha$ ,  $\beta$ ].DM  
 $\alpha$ : Start address  $\alpha < \beta$   
 $\beta$ : End address  $\alpha > \beta$  will result in error

Function : Dumps the contents of the coverage memory.

Description:

There are two types of coverages, PC (Program Counter) and DATA.

- o For PC coverage, the number of executions for each program address in the specified range will be counted. The number of counts can be recorded from 0 to FFH. A count which is greater than FFH is indicated as FFH.
- o For data coverage, the data memory status (write condition). The meaning of each indication is as indicated below:

Indication	Meaning
-----	... Indicates a bit which has never been written
*	... Indicates a bit to which 0 and 1 have been written.
0	... Indicates a bit to which only 0 has been written.
1	... Indicates a bit to which only 1 has been written.

Example 1 : To display the contents of PC coverage:  
If a and b are not specified, the contents of address 0 to 7FH will be displayed.

```
BRK>.DM$$
PC (1) / DATA (0) COVERAGE : ? 1↓
ADDR 0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Example 2: To display the contents of data coverage:

If  $\alpha$  and  $\beta$  are not specified, the contents of address to 3FH will be displayed.

RF (register file) will be excluded from the coverage.

```
BRK>.DM$$
PC (1) / DATA (0)  COVERAGE : ? 0↓
ADDR 0/8  1/9  2/A  3/B  4/C  5/D  6/E  7/F
0000 ----  ----  ----  ----  ----  ----  ----  ----
0008 ----  ----  ----  ----  ----  ----  ----  ----
0010 ----  ----  ----  ----  ----  ----  ----  ----
0018 ----  ----  ----  ----  ----  ----  ----  ----
0020 ----  ----  ----  ----  ----  ----  ----  ----
0028 ----  ----  ----  ----  ----  ----  ----  ----
0030 ----  ----  ----  ----  ----  ----  ----  ----
0038 ----  ----  ----  ----  ----  ----  ----  ----
```

Note: In some models, if  $\alpha$  and  $\beta$  are not specified, an error may occur.

5.3.6 Program pattern generator (PPG) control commands

- (1) Initialize PPG data  
IG (Initialize PPG ..... Data)
- (2) Change PPG data  
CG (Change PPG ..... Data)
- (3) Dump PPG data  
DG (Dump PPG ..... Data)
- (4) Execute/stop PPG, set PPG operation mode  
EG (Execute PPG)
- (5) Save PPG data  
SG (Save PPG ..... Data)
- (6) Load PPG data  
LG (Load PPG ..... Data)
- (7) Verify PPG data  
VG (Verify PPG ..... Data)

<code>.IG</code> Initialize PPG data
--------------------------------------

Format    :  $[\alpha, \beta, \gamma].IG$

$\alpha$ : Start address (0 to 1FFF)     $\alpha < \beta$

$\beta$ : End address (0 to 1FFF)       $\alpha > \beta$  will result in error

$\gamma$ : Data (0 to FFFF)

Function : Initializes the data in the address range  $\alpha - \beta$  to Data  $\gamma$ .

When clearing all PPG data (0 to 1FFF) to 0s, it is not necessary to specify  $\alpha$ ,  $\beta$ , and  $\gamma$ .

Example 1: To clear all PPG data to 0s:

```
BRK> .IG$$
```

Example 2: To initialize PPG data in the range from address 0 to address FFH to 5555:

```
BRK> 0,FF,555.IG$$
```

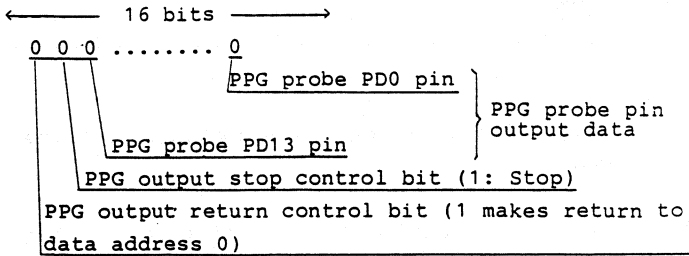
Note      : .IG input will not be accepted during PPG execution.





## UM IE-17K

- o PPG data has the following meanings:



- o When specifying the PPG stop address, set 2 successive stop control bits to 1.
- o If .CG is executed during PPG execution, PPG will stop.
- o PPG data set to address 0H will be output for a period of twice the specified step rate.
- o When setting the stop control bit, the interval between the PPG execution start point and the first stop point must be at least 3 $\mu$ s.

.DG Dump PPG data

Format : [ $\alpha$ ],b.DG

$\alpha$ : Start address (0 to 1FFF)  $\alpha \leq \beta$

$\beta$ : End address (0 to 1FFF)  $\alpha > \beta$  will result in error

Function : Displays PPG data from address  $\alpha$  to address  $\beta$ .

Example : To dump PPG data in the address range 0 - 10H:

BRK>0,10.DG\$\$

0000 : 0000000000000000

0001 : 0000000000000000

0002 : 0000000000000000

0003 : 0000000000000000

0004 : 0000000000000000

0005 : 0000000000000000

0006 : 0000000000000000

0007 : 0000000000000000

0008 : 0000000000000000

0009 : 0000000000000000

000A : 0000000000000000

000B : 0000000000000000

000C : 0000000000000000

000D : 0000000000000000

000E : 0000000000000000

000F : 0000000000000000

0010 : 0000000000000000

Note : If .IG is executed during a PPG execution, the PPG execution will stop.

<code>.EG</code> Execute/stop PPG, set PPG operation mode
---

Format    : .EG

- Function : (1) Executes PPG.  
          (2) Stops PPG.  
          (3) Specifies the effective bit, step rate, as the PPG operation mode.

### Setting the effective bit

- o When it is necessary to fix pin outputs PD0 - PD13 to 0 (low level), set the select bit to 0. To output as is, set it to 1. The following shows how the select bits and the pins of the PPG probe:

```
SELECT BIT: 001.....1
              |
              | PD0 pin
              | PD13 pin
              |
              Fix to 0
```

- o Step rate setting specifies the time taken to output 1 step of PPG data. It can be set from 1us to 13333us (decimal). If 0 is selected as the step rate, 1 step will be 13333us.

Example : To fix output pins 13 and 1 of the PPG to 0, and the execution speed per step to 10ms/step:

```
BRK> .EG$$
PPG    RUN(1)/RESET(2)/SELECT(3):?3↓
SELECT BIT: 0001111111111101↓
STEP RATE : 10000↓
```

- Note :
- o PPG data set to address 0H will be output for a time period of twice the specified step rate.
  - o If a RESET is executed during execution, the PPG will stop and the effective bits specified by the SELECT BIT will be set to high level.

- o If RUN is executed again during RUN, RUN will be continuously executed.
- o If SELECT is specified during RUN, the PPG will stop.
- o When PPG is stopped by the stop control bit, the effective PPG data of the address at which the stop control bit was set will be output.
- o The following formula gives an accurate execution speed per step:

$$\text{(Execution speed)} = \frac{1}{4.9152 \times 10^6} \times \text{INT}[4.9152 \times (\text{step rate})]^*$$

\* INT[ ] is the maximum integer value which does not exceed [ ].



.LG0	.LG1	Load PPG data
------	------	---------------

Format : { .LG0 }  
          { .LG1 }

RS-232C Channel 0: LG0  
          Channel 1: LG1

Function : Inputs the PPG data from the RS-232C channel specified  
          by .LG0 or .LG1.

Example : To input the PPG data from channel 0:

BRK> .LG0\$\$

.VG0 .VG1 Verify PPG data

Format : { .VG0 }  
          { .VG1 }

RS-232C Channel 0: VG0  
          Channel 1: VG1

Function : Verifies the PPG data against the Intel HEX format data sent from the RS-232C line specified by .VG0 or .VG1. If they coincide, "Verify OK" will be displayed. If they do not coincide, "Verify NG" will be displayed.

Example : To verify the PPG data input from RS-232C channel 0:

BRK> .VG0\$\$  
Verify OK



## 5.3.7 Help command

(1) Lists all commands (Help)

.H            Lists all commands (Help)
---

Format     : .H

Function   : Lists all commands.

Example    : To list all commands:

```
BRK>.H$$  
.IP .CP .DP .FP .SP0 .SP1 .LP0 .LP1 .VP0 .VP1 .XS0 .XS1  
          << PROGRAM MEMORY COMMAND >>  
.ID .CD .DD .D  
          << DATA MEMORY COMMAND >>  
.R .RN .BG .BK .CA .S  
          << EMULATION COMMAND >>  
.CC .CT .DC .DT .DM .SC0 .SC1 .LC0 .LC1 .VC0 .VC1  
          << BREAK , TRACE CONDITION COMMAND >>  
.IG .CG .DG .EG .SG0 .SG1 .LG0 .LG1 .VG0 .VG1  
          << PULSE GENERATER COMMAND >>
```

## 5.3.8 Other commands

- (1) Define macro  
U (Define macro)
- (2) Execute macro  
M (Execute macro)
- (3) Dump macro  
=C (Dump macro)
- (4) Loop  
< > (Loop)

U	Define macro
---	--------------

Format : U $\alpha$  command string  
 $\alpha$ : Macro name (0 to 9, A to Z)

Function : Defines macro. Macro name is expressed in one digit from 0 to 9 or one character from A to Z.

Example : To define macro Z:

BRK>UZ.BK80.DD\$\$

This defines .BK80.DD as macro Z.

M	Execute macro
---	---------------

Format : M $\alpha$   
 $\alpha$ : Macro name (0 to 9, A to Z)

Function : Executes macro defined by command U. Macro name is expressed in one digit from 0 to 9 or one character from A to Z.

Example : To defined macro which will break program execution, and dumps the contents of the register file, and execute it:

```
DMA > .RSS
BRK > U1 BK80 .DDSS
BRK > .RNSS
DMA >
RUN >
DMA > M1SS
ADDR INSTRUCTION
1800 00000 BREAK
1800 00000 OVERRUN
001D 074F0 NEXT
PC SP AR WR BR MP IX
001D 5 0000 1 0 *** 070
PSW : DB CP CY Z IXE MPE JG
      0 0 0 0 0 0 0
RP 0123456789ABCDEF
*6 4080000000000000

0080 : 2 5 2 3 4 5 6 1 7 A A B C D E 1
0090 : 0 0 0 3 4 0 6 0 0 9 A B C D E 0
00A0 : 0 1 2 3 4 5 6 1 0 9 A B C D E 0
00B0 : 0 0 2 3 4 7 7 F 0 0 A B C D E 2
```

=C      Dump macro
--------------------

Format    : =C $\alpha$

$\alpha$ : Macro name (0 to 9, A to Z)

Function : Dumps macro specified by  $\alpha$ . Macro name is expressed in one digit from 0 to 9 or one character from A to Z.

Example  : To define macro 2 as ".R.RN" and dump its contents:

BRK>U2.R.RN\$\$

BRK>=C2\$\$R.RN

```
< > Loop
```

Format :  $\alpha$ <command string>  
 $\alpha$ : Number of loops

Function : Repeatedly executes the command string specified by < > for  $\alpha$  times.

Example : To repeat step and dump twice:

```
BRK>2<1.S0.7F.DD>$$  
BR RP PC INSTRUCTION  
0 *6 001D 074F0 ↓  
  
0000:0 0 B 8 6 0 0 0 0 0 0 0 0 0 0 0 0  
0010:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0020:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0030:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0040:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0050:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0060:4 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0  
0070:4 0 F 0 0 0 0 0 0 1 0 0 7 0 0 C 0
```

```
BR RP PC INSTRUCTION  
0 *6 001E 10021 ↓  
  
0000:0 0 B 8 6 0 0 0 0 0 0 0 0 0 0 0 0  
0010:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0020:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0030:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0040:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0050:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0060:4 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0  
0070:4 0 F 0 0 0 0 0 0 1 0 0 7 0 0 C 0
```

BRK>

- Note: o When the number of loops  $\alpha$  is not specified, the number  $2^{32}-1$  will be selected.  
o To prevent errors, each command used in the command string must be written without abbreviating.





## Chapter 6 Programmable Pulse Generator (PPG)

The IE-17K has a 14-bit parallel PPG function. The output pattern can be set up to 8192 steps and the step rate can be specified approximately from 1us to 13333us in 1us steps.

### 6.1 Displaying, Modifying PPG Data

PPG data can be displayed or changed using the .DG command or the .CG command. When the .DG or .CG command is executed, the PPG stops its operation.

Example 1: To display the PPG data from address 0 to address 3:

```
BRK>0.3.DG$$
0000 : 0000000000000000
0001 : 0000000000000000
0002 : 0000000000000000
0003 : 0000000000000000
BRK>
```

Example 2: To change the PPG data of address 0 to '0011111111111111':

```
BRK>.CG$$
0000 : 0000000000000000 - 0011111111111111↓
0001 : 0000000000000000 - 3
BRK>
```

### 6.2 Setting the Step Rate

The step rate can be changed by selecting SELECT(3) of the .EG command. When the SELECT(3) is selected the PPG stops.

Each of the 14 output pins, can be individually specified as effective or not effective. Pins specified as effective bits output high levels when the PPG is stopped, and output PPG data when the PPG is in operation.

Pins not specified as effective bits outputs low levels regardless of the condition of the PPG data.

The step rate can be specified approximately from 1us to 13333us in 1us steps.

One step is approximately 1us. However, if the step rate is set to a small value, 1 step may be shorter than 1us.

**Example** : To select bits 0 to 7 as effective, and set the step rate to 100US/step:

```
BRK > .EGSS
PSG RUN(1) / RESET(2) / SELECT(3) : ? 3 ↓
SELECT BIT : 0000000011111111 ↓
STEP RATE : 100 ↓
BRK >
```

### 6.3 Executing PPG, Stopping PPG

The PPG execution can be started/stopped using the .EG command. PPG execution can be started only when the PPG is in the stop state.

Example 1: To start PPG execution:

```
BRK > .EGSS  
PSG RUN(1) / RESET(2) / SELECT(3) : ? 1↓  
BRK >
```

Example 2: To stop PPG execution:

```
BRK > .EGSS  
PSG RUN(1) / RESET(2) / SELECT(3) : ? 2↓  
BRK >
```

## UM IE-17K

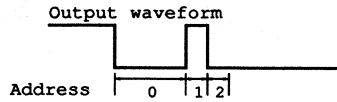
### 6.4 Notes On Using the PPG

PPG data set to address 0H will be output for a period of twice the specified step rate.

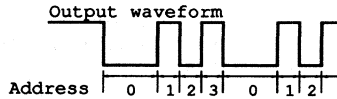
When stopping the PPG using the PPG stop control bit, set 2 successive stop control bits to 1.

#### Example:

```
0000 : 00000000000000000000
0001 : 00111111111111111111
0002 : 01000000000000000000
0003 : 01000000000000000000
```



```
0000 : 00000000000000000000
0001 : 00111111111111111111
0002 : 00000000000000000000
0003 : 10111111111111111111
```

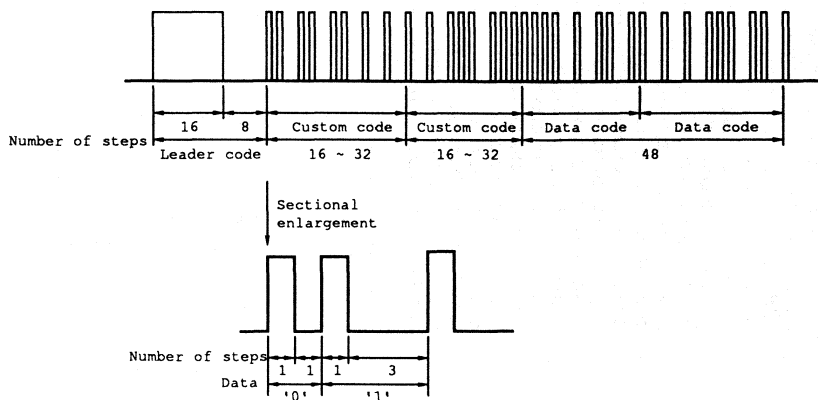


### 6.5 PPG Application Example

The output data from the uPD6122G remote control transmitter IC can be simulated by the PPG. When the step rate is set to 563, 106 to 138 steps of data can be transmitted.

The following figures show the output wave-form of the uPD6122G and a part of the pulse generator data.

Example of output wave-form





## Chapter 7 Program Execution

The program can be executed in one the following ways:

- (1) Real-time emulation
- (2) Single-step emulation

### 7.1 Real-Time Emulation

The .RN command is used to execute the program with the same speed as the actual product. Break can be generated with arbitrary condition by setting break point. Execution can be regardlessly stopped by executing the .BK command.

Example 1: To execute after resetting the CPU:

```
BRK > .R$$  
BRK > .RN$$  
RUN >
```

Example 2: To resume after interrupting real-time emulation:

```
RUN > .BK$$  
ADDR INSTRUCTION  
0027 1E7F2 BREAK  
0028 0C026 OVERRUN  
0029 070E0 NEXT  
PC SP AR WR BR MP IX  
00F2 0 0000 * * *** ***  
PSW : DB CP CY Z IXE MPE JG  
0 1 0 1 * * 0  
RP 0123456789ABCDEF  
*0 8D98D99999FFAD9D  
  
BRK > .RN$$  
RUN >
```

## 7.2 Setting Break Points

By setting break points, break can be generated with arbitrary condition. Break conditions can be program memory address, data memory address, writing data to the data memory, change of the logic analyzer probe pin status, etc. Break conditions can be set so that the program execution can be stopped when two or more break conditions are simultaneously satisfied or when two or more break conditions are successively satisfied.

**Example:** To generate break when the program counter address becomes 00F0H:

```
BRK > _CCSS
A) LEVEL (1, 2) : ? 1 ↓
B) UNIT (0 - 3) : ? 0 ↓
   CATG (C - L) : ? C ↓
C) CONDITION AND(1) / OR(0) : ? ↓
D) PROG ADDR UPPER : FFFF ? 00F0 ↓
   PROG ADDR LOWER : 0000 ? 00F0 ↓
   MATCH(1) / UNMATCH(0) : 0 ? 1 ↓
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : 0 ? 2
```

```
BRK > _CCSS
A) LEVEL(1, 2) : ? 2 ↓
B) LEVEL2 : 0123
   DEPTH-3 : 1101 ? ↓
   DEPTH-2 : 1101 ? ↓
   DEPTH-1 : 1101 ? ↓
   DEPTH-0 : 1101 ? 1000 ↓
   INITIAL DEPTH : 0 ? ↓
```

```
BRK > _RSS
BRK > _RNS
BRK > _BKSS
ADDR INSTRUCTION
00F0 1D069 BREAK
00F1 1D059 OVERRUN
00F2 1D045 NEXT
PC SP AR WR BR MP IX
00F2 0 0000 * * *** ***
PSW : DB CP CY Z IXE MPE JG
      0 0 0 0 * * 0
RP 0123456789ABCDEF
*0 0000099990008005
```

```
BRK >
```



### 7.3 Single-Step Emulation

This mode is used to check the process flow by executing the program one step at a time.

Example 1: To execute one instruction:

```
BRK> .SS$  
BR RP PC INSTRUCTION  
* *0 0034 0C03D  SS$  
  
BRK>
```

The instruction displayed at address 0034H has not been executed.

The number of steps can be specified by specifying a numeric value before the .S command.

Example 2: To execute two instructions at addresses 33H and 34H:

```
BRK>33.CAS2.SS$  
BR RP PC INSTRUCTION  
* *0 0034 0C03D  
* *0 003D 11001  SS$  
  
BRK>
```

The next instruction can be executed by inputting a space after executing the .S command.

Example 3: To execute two instructions at addresses 33H and 34H:

```
BRK>33.CAS.SS$  
BR RP PC INSTRUCTION  
* *0 0034 0C03D  
* *0 003D 11001  SS$  
  
BRK>
```

..... Inputting a space  
executes one step.



## Chapter 8 Programming the PROM for the SE Board

The PROM file format HEX code of the program (modified by the IE-17K) which is output from the AS17K can be output to line 0 or 1. The PROM can be programmed for the SE board by connecting the PROM programmer to line 1.



## Chapter 9 Error Messages

The IE-17K generates an error message when a command is incorrectly input or a hardware problem occurs.

### 9.1 Error Messages Related to Commands

A command error message is displayed when the command name is incorrectly input or the number of arguments are not correct. The following lists the error messages related to commands.

- (1) ?MLA MISSING <  
This message is displayed when the number of '<'s is fewer than the number of '>'s.
- (2) ?MRA MISSING >  
This message is displayed when the number of '>'s is fewer than the number of '<'s.
- (3) ?MLP MISSING (  
This message is displayed when the number of '('s is fewer than the number of ')'s.
- (4) ?MLP MISSING )  
This message is displayed when the number of ')'s is fewer than the number of '('s.
- (5) ?MNF MACRO COMMAND NOT FOUND  
This message is displayed when a character string which begins with '.' does not exist as a macro command.
- (6) ?NVQ NO VALUE IN Q-REGISTER  
This message is displayed when an attempt is made to execute the contents of the Q register as a macro when the Q register contains nothing.
- (7) ?SYN INVALID SYNTAX  
This message is displayed when an syntax error other than indicated in 1 - 6 above is found.

- (8) ?FAP FAIL TO ACCESS PSG  
This message is displayed when a verify error is generated when writing pulse patter generator data.
- (9) ?IPE INPUT ERROR  
This message is displayed when an invalid value is set for the .CC command.
- (10) ?INA ILLEGAL NUMBER OF ARGUMENTS  
This message is displayed when the number of arguments for the macro command is insufficient.
- (11) ?IVA INVALID ARGUMENT  
This message is displayed when the argument value is illegal.
- (12) ?POS INVALID ADDRESS  
This message is displayed when an address which exceeds the program memory address range of the product is specified.
- (13) ?RSE CPU RESET ERROR  
This message is displayed when an attempt is made to execute a command which should not be carried out during emulation of the program.
- (14) ?RNE CPU RUN ERROR  
This message is displayed when an attempt is made to execute a command which should not be carried out during emulation of the program.
- (15) ?RTE RUN ERROR  
This message is displayed when an attempt is made to execute a command which should not be carried out during emulation of the program.
- (16) ?WRE WRITE ERROR  
This message is displayed when a verify error occurs when writing to the memory.

### 9.2 Hardware Error

A hardware error is displayed when the IE-17K malfunctions during program execution. The following describes these hardware error messages.

(1) SYSTEM REGISTER ACCESS ERROR

This message is displayed when an attempt is made to access bit which is not mounted in system registers, but is located in the AR register.

(2) STACKOVER/UNDER FLOW

This message is displayed when the stack pointer overflows or underflows.

(3) RAM NOT INITIALIZE

This message is displayed when an instruction to read data memory is executed a data memory (except port) to which nothing has been written or an initial value has not been determined.

(4) ILLEGAL RAM WRITE

This message is displayed when an attempt is made to write to an data memory which does not exist.

(5) ?IOS INVALID OPTION SWITCH AT 0000

This message is displayed when the option switch specification differs from that of the option switch on the SE board, when the program is loaded to, or executed by the IE-17K or the program is executed.

(6) ?ISE INVALID SE BOARD NUMBER [ 00 - 00 ]

This message is displayed when the device file used for the assembler differs from the SE board when the program is loaded to executed by the IE-17K or the program is executed. This message may also be displayed when the SE board is not properly installed. The left side number indicates the SE board number, and the right side number indicates the number contained in the device file.

(7) ?IDI INVALID DEVICE ID NUMBER [ 00 - 00 ]

This message is displayed when the device file used for the assembler differs from the device on the SE board when the program is loaded to the IE-17K.

This message may also be displayed when the device is not properly mounted on the SE board. The left side number indicates the device number on the SE board, and the right side number indicates the number contained in the device file.

(8) --- NO SWITCH OPTION ---

This message is displayed when the option information is not successfully loaded during loading of the program to the IE-17K.

(9) PC ERROR!

This message is displayed when the program counter does not operate due to a malfunction on the SE board.

The following error messages (10 - 18) will be displayed when an error is detected during execution of the self-diagnostic test. The test performed by the IE-17K upon power on or reset. All of these messages indicates hardware malfunction which requires immediate repair.

(10) MEMORY ERROR --> 0000:0000 - 7000:FFFF

This message is displayed when the memory used by the IE-17K malfunctions.

(11) MEMORY ERROR --> E000:0000 - E000:FFFF

This message is displayed when the memory used by the pulse generator malfunctions.

(12) DEVICE ERROR --> PTC ( UPD71054 ) #0

This message is displayed when programmable timer 0 (uPD71054) malfunctions.



- (13) DEVICE ERROR --> PTC ( UPD71054 ) #1  
This message is displayed when programmable timer 1 (uPD71054) malfunctions.
- (14) DEVICE ERROR --> PIU ( UPD71055 ) #0  
This message is displayed when parallel interface 0 (uPD71055) malfunctions.
- (15) DEVICE ERROR --> PIU ( UPD71055 ) #1  
This message is displayed when parallel interface 1 (uPD71055) malfunctions.
- (16) DEVICE ERROR --> SCU ( UPD71051 ) #0  
This message is displayed when serial control unit 0 (uPD71051) malfunctions.
- (17) DEVICE ERROR --> SCU ( UPD71051 ) #1  
This message is displayed when serial control unit 1 (uPD71051) malfunctions.
- (18) DEVICE ERROR --> ICU ( UPD71059 )  
This message is displayed when the interrupt controller (uPD71059) malfunctions.

The following messages (19) to (24) will be displayed when the CPU of the IE-17K malfunctions, e.g., CPU runaway.

- (19) << DIVIDE BY ZERO >>  
When division by zero is attempted.
- (20) << CHECK FIELD >>  
When memory boundary is exceeded.
- (21) << SINGLE STEP >>  
When a single step is executed.
- (22) << BREAK MODE >>  
When a break instruction is executed.

(23) << OVERFLOW >>

When an overflow has occurred during operation.

(24) << NMI >>

When NMI is generated.

# NEC Customer Services

## NEC's Commitment to Information

Our offices throughout Europe are always at your service for comprehensive support. Here are some of the technical services we provide:

- INSECT
- Seminars
- Update service
- Hotline
- Mailing list
- University program

## INSECT

**I**nformation **S**ystem and **E**lectronic **C**atalog.

This is an on-line information service. Via a telephone link you can call up the latest data on all VLSI devices available from NEC. This includes enhancements, news and the most recent application know-how.

The service is free to our customers and other interested parties. For a menu-driven guest session, you can dial in to INSECT via the international packet switching network - in Germany this is DATEX-P - using of these numbers  
45 21 10 13 020/030  
and responding to the request for USERNAME and PASSWORD simply with "customer".

## Seminars

No-one is more aware than NEC of the difference that a brief intensive training course can make to your mastery of advanced and often complex devices. We hold regular workshops and seminars at local NEC offices, in our Düsseldorf headquarters, or on customer premises. For information on NEC workshops and seminars, please contact your nearest office.

## Update Service

When you buy an evaluation package from NEC, you become automatically entitled to one year's free updates for both hardware and software. All updates reach you fast and reliably via a courier service. In a field where rapid changes are the norm, you can be thus be sure of working with the most up-to-date development tools.

## Hotlines

NEC's offices located throughout Europe are responsible for technical support and customer services. On the back cover of this brochure you can check which office is most convenient for you to contact. You are also welcome to contact our European headquarters in Düsseldorf directly.

## Mailing list

Our engineering staff produces frequent additions to the available technical documentation in the form of application notes, product news and technical letters. If you would like to be included on our mailing list for this documentation, please inform us.

## University Program

Many of our products, because of their complexity and dedicated application support through EB tools, are interesting subjects for graduate studies. NEC is always ready to discuss this possibility.