JRH MASTER COPY

DISK MACRO
ASSEMBLER/TEXT EDITOR (MAE)
FOR 6502 ATARI COMPUTERS

CONTENTS


```
**********************************************************
*                                                        *
*  WRITTEN ENTIRELY IN MACHINE LANGUAGE BY               *
*  EASTERN HOUSE SOFTWARE, SERIAL #:                     *
*                                                        *
*                                                        *
*                                                        *
**********************************************************
```

COPYRIGHT NOTES

This manual and the object code is serial numbered and protected by a legitimate copyright. No part of this manual may be copied or reproduced without the express written permission of the owner, Carl Moser. It is a Federal crime to make a copy of the manual or diskette for use by anyone other than the individual who purchased this software or the individual a company purchased the software for.

Thus, you are in violation of Federal Copyright Laws if you do one of the following:

- Make a copy of the manual.
- If you allow someone else to use your copy of the object media while you retain a copy or are using a copy.
- If you, your company, or others purchase one or more copies and more individuals simultaneously use this software than the number purchased.
- If you allow someone else to do the copying of this material, you will be considered as a party to the infringement.

A reward will be provided for anyone who supplies information which leads to the prosecution of parties who violate this copyright.

We do not presume that you are or will violate copyright laws. Most users do not. Some though do, and may not realize the consequences for violation of this Federal Law. Penalities and fines can be quite severe for both individuals and companies who infringe this copyright.

Most importantly, software houses like the one which wrote this software have incurred a tremendous investment that can not be fully recovered if current illegal copying continues. Also, updates and program maintenance will have to be terminated if the return on investment is not sufficient.

If (for whatever reason) your diskette or cassette becomes defective, EHS will exchange it for a small charge.

Finally, an expressed appreciation is given to the purchaser of this software. We hope that you find it a valuable and worthwhile investment.

If you encounter any problems, contact us at:

Eastern House Software

Carl Moser
3239 Linda Drive
Winston-Salem, N. C. 27106

or

J. R. Hall
4145 Transou Road
Pfafftown, N. C. 27040

1. INTRODUCTION

The MAE is a highly sophisticated software package with many powerful commands. Most commands are easy to use and understand. However, it is not always possible to provide an exact example due to the complexity of the operation. It is also difficult to clearly get across some of the most powerful assembler concepts. Therefore, we have provided numerous examples in this manual as well as example disk files on the supplied diskette. In order to get the most out of this manual, read PARTS 1 to 7 and study the examples in PART 8 before going to PART 9 (Getting Started With MAE). Although you may not totally understand all of the MAE commands when you get to PART 9, most commands will become straight foward once you have a chance to work with the MAE. Finally, in PART 10, there is a word processor for your use in writing letters, books, or anything. In fact, this manual was written using the word processor. Now let's get started learning about the software !!!

The Macro Assembler (ASSM), Text Editor (TED), and Machine Language Monitor (MLM) resides simultaneously with the disk operating system (DOS) in less than 21K bytes of memory. The collective assembler and text editor is referred to as MAE. MAE was designed to work with the ATARI 400 or 800 with at least 32K of memory and at least one ATARI 810 disk drive. The included MLM and DOS may be used for interfacing the cassette and disk as desired by the user. In addition, the MLM provides 25 commands useful in debugging the assembled object code.

Some unique features of MAE are:

- Macro, Conditional Assembly, and Interactive Assembly.
- Labels up to 31 characters in length.
- Auto line numbering for ease of text entry.
- Creates both executable code in memory and relocatable object code on disk.
- Word processing feature for composing letters and other text.
- Loading and storing via disk.
- Supports ATARI printer.
- String search and replace capability, plus other powerful editing commands.

As mentioned, the total object code occupies less than 21K of memory. In addition to this, sufficient memory must be allocated for the text file and label file (symbol table). Approximately 5K is sufficient memory for the text file for small programs or larger programs if assembled from disk. If an executable object code file is to be stored in memory during assembly, sufficient memory must be provided for that also. On cold start entry, MAE will set the file boundries as follows:

- Text File = $6800-$7C1C
- Label File = $2680-$2FFC
- Relocatable Object Buffer = $5800

In a 32K ATARI system, these boundaries leave practically no memory for object code storage (see Memory Map - PART 12). Therefore, we recommend 48K of memory for more useability. Whether you use 32K, 40K, or 48K of memory, the BASIC cartridge should not be installed.

The label file and text file that MAE generates is position independent and may be located practically anywhere in RAM memory (see .SE command). The object code file location is dependent on the beginning of assembly (.BA pseudo op) and the .MC pseudo op.

MAE was designed such that records in the label file and text file are variable in length and directly dependent on the number of characters to be stored. This results in more efficient utilization of memory.

Initial entry (or cold start) to MAE is via a special MLM command (see AC command). Initial entry provides the following default parameters:

- Format = set
- Manuscript = clear
- Auto line numbering = off
- Text file and Label file = clear

MAE uses a prompter character (]) to indicate that it is ready to accept commands. Command mnemonics referenced in this document are printed with the prompter (example]BR). When inputting a command, you should not type "]" preceeding the mnemonic.

This software has been extensively tested and is believed to be entirely reliable. It would be foolish to guarantee a program of this size and complexity to be free of errors. Therefore, we assume no responsibility for the failure of this software. We will take any reasonable steps to fix any problems with this software. If you do find a problem, please feel free to write us describing in detail the problem.

2. FILES CONTAINED ON THE DISKETTE

The supplied diskette contains the following files. As time goes by, new files will be added in order for you to get the most out of the ATARI MAE.

Filename	Description
DOS.SYS	DOS, MLM, and MAE Software
WORDP.EXE	STP Word Processor object code
WORDP.INS	Example of raw text for word processor
MEMTST.EXE	Memory Test object code
MEMTST.INS	Memory Test instructions
MAE.NOT	Some notes on the MAE
EXAMP.xx	Example source programs for use with MAE

3. MACHINE LANGUAGE MONITOR (MLM) COMMANDS

The MLM provides 25 commands which are most useful to the machine language programmer. It provides the user with the capability to easily interact with the 6502 microprocessor and system memory. Used with MAE, it provides the powerful flexibility to assemble programs and then de-bug the machine code all within one software package. The MLM uses the ATARI screen editing capability. This feature makes the MLM powerful and easy to use.

The following is a list of the MLM commands. Carefully read over the commands and examples. Then practice with the MLM to gain a better insight into their use.

. -- COMMAND PROMPT

A period is used to indicate the MLM is ready for a command.

M XXXX YYYY -- DISPLAY MEMORY

Display memory starting at hex address XXXX and ending at YYYY.

Example -

```
M 6531 653F
 :6531 01 02 03 04 05 06 07 08
 :6539 09 0A 0B 0C 0D 0E 0F 10
```

Note -- If only start address is entered, 24 memory locations will be displayed.

Note -- For long memory displays, the control-1 key can be use to stop and start the listing.

Note -- To abort a long listing, press the space bar.

I XXXX YYYY -- INTERROGATE MEMORY Interrogate memory starting at hex address XXXX and ending at YYYY. The interrogate command works just like the .M command except it also displays the ASCII equivalent of the memory contents. All cursor control codes are displayed with a question mark (?).

R -- DISPLAY REGISTERS

Display 6502 registers.

Example Printout -

```
*  PC  AR XR YR PR SP
;  7013 41 11 FA 03 FA
```

PC = program counter; AR = accumulator; XR = X register
YR = Y register; PR = status register; SP = stack pointer

: -- ALTER MEMORY

Indicates that the following hex address and line of hex data will be used to alter memory. Cursur up and over to location and change bytes -- press RETURN.

; -- ALTER 6502 REGISTERS

Used to modify 6502 registers. Cursor up and over to register and change bytes -- press RETURN.

G XXXX -- GOTO

GOTO address specified by XXXX and execute program. Program must contain a BRK instruction to return to MLM. If XXXX is not given, the GOTO address defaults to the program counter.

C XXXX -- CHANGE MEMORY

This is a special mode to alter memory. The address specified by XXXX is the starting hex address.

Example - .C 6000

SCREEN DISPLAYS - 6000 YY
ENTER HEX DATA AT YY
AND PRESS RETURN

SCREEN DISPLAYS - 6001
ETC

To exit the change mode, press RETURN instead of entering data. The prompter (.) will be displayed.

X -- EXIT

Exit the monitor and return to DOS menu.

S XXXX YYYY -- SAVE MEMORY TO CASSETTE

Save memory starting at hex address XXXX to ending address YYYY.

Note -- The end address must be the actual address+1.

L XXXX -- LOAD MEMORY FROM CASSETTE

Load memory from cassette and store starting at hex address XXXX. Binary data saved using the S XXXX YYYY command can be loaded into memory at any location (as defined by L XXXX).

? -- ERROR

A question mark will be printed if a bad command or bad hex data is entered. It will also be given if any command tries to alter a ROM or non-existent memory location.

F XXXX YYYY ZZ -- FILL MEMORY

Fill memory starting at hex address XXXX to address YYYY with the hex character ZZ.

Example - F 1000 10F0 E3 - Fill memory from $1000 to $10F0 with $E3.

H XXXX YYYY 'ZZZZZ -- HUNT FOR ASCII STRING

Hunt memory from XXXX to YYYY for the ASCII string ZZZZZ

Example - H 1700 2A80 'ATARI COMPUTER -- Hunt memory from $1700 to $2A80 for the ASCII string ATARI COMPUTER .

Note - The ASCII string can be up to 20 characters long.

Note - If a match of the ASCII string is found, the hex address will be listed to the screen. If no match is found, only the command prompt (.) will be displayed.

H XXXX YYYY ZZ ZZ ZZ ZZ -- HUNT FOR HEX CHARACTERS

Hunt memory from XXXX to YYYY for the hex characters ZZ ZZ ZZ ZZ.

Example - H 1700 2A80 20 00 07 A9 FF -- Hunt memory from $1700 to $2A80 for the hex characters 20 00 07 A9 FF .

Note - The hex characters can be up to 20 hex bytes long.

Note - If a match of the hex characters is found, the hex address will be listed to the screen. If no match is found, only the command prompt (.) will be displayed.

T XXXX YYYY ZZZZ -- TRANSFER MEMORY

Transfer memory from address XXXX to YYYY and store it starting at address ZZZZ.

Example - T 5000 5100 C000 - Transfer memory from $5000 to $5100 and store it starting at $C000 .

K XXXX YYYY ZZZZ - COMPARE MEMORY

Compare memory from address XXXX to YYYY with the memory starting at ZZZZ.

Example - K 5000 5100 C000 - Compare memory from $5000 to $5100 with the memory starting at $C000 If any memory locations are different, the address will be printed on the screen.

D XXXX - DISASSEMBLE MEMORY

Disassemble memory starting at hex address XXXX.

Example - D A000 - Disassemble memory starting at $A000. The screen will clear and display the hex code as well as the disassembled mnemonics. The control-1 key is used to stop and start the listing. To terminate the listing, press the space bar.

```
,A000  A5  CA    .LDA $CA
,A002  D0  04    .BNE $A008
,A004  A5  08    .LDA $08
,A006  D0  45    .BNE $A04D
 etc
```

Note - When an unimplimented opcode is encountered, the mnemonic field will display ???.

, - ALTER DISASSEMBLE LISTING

A comma command is used to alter the hex code printed out by the disassemble command. After the listing has been stopped with space bar, simply cursor up and over and change hex code (up to 3 memory locations can be modified). When the RETURN key is pressed, the disassembly process will begin again.

B XXXX YYYY - CALCULATE BRANCH

Calculate the branch value from address XXXX to YYYY.

Example - B 4000 4013 - Calculate the value of a branch instruction when the program counter is at $4000 and branch to instruction is at $4013. In this case, the hex value 13 will be displayed.

XXXXX - CONVERT DECIMAL TO HEXIDECIMAL

Convert the decimal number XXXXX to a hexidecimal number. The maximum decimal is 65535.

Examples - #65535 , #1024 , #16 , $8

$ XXXX - CONVERT HEXIDECIMAL TO DECIMAL

Convert the hexidecimal number XXXX to a decimal number. The maximum hexidecimal number is FFFF.

Examples - $FFFF , $E034 , $A00 , $FF , $B

PS -- PRINTER SET

Set the MLM so that all characters sent to the screen are also sent to the ATARI printer.

PC -- PRINTER CLEAR

Stop sending characters to printer.

AC -- MAE COLD START

Enter MAE and set all parameters to their default values.

AW -- MAE WARM START

Enter MAE and keep all parameters at their current values.

ADDITIONAL NOTES

(1) Spaces have been shown in the examples of the commands. Spaces are optional and are not required by the monitor.

(2) Error messages associated with the printer, disk or cassette will be printed to the screen in the form 'SYSTEM ERROR=XXXXX' where XXXXX is a decimal number which represents the normal BASIC error numbers (see BASIC and/or DOS manuals).

4. TEXT EDITOR (TED) FEATURES

The TED occupies approximately one-half the total memory space of the MAE software. The purpose of the TED is to setup and maintain the source file by interacting with the user via various commands.

When inputting to the TED, the user has available the full capabilities of the built in cursor-oriented screen editor. When listing to the CRT or printer, the user has control of the output via the following keys:

Control S - Temporarily halt outputting and await input of one of the following keys.
Control Z - Return to prompter "]" level.
Control O - Continue processing but suppress output except for errors.
Control Q - Continue outputting after tempory halt (control S).

A. Commands

The TED provides 27 command functions. Each command mnemonic must begin immediately after the prompter (]). When entered, a command is not executed until the RETURN key is depressed. Although a command mnemonic such as]PR may be several non-space characters in length, MAE only considers the first two. For example,]PR,]PRI,]PRINT, and]PRETTY will be interpreted as the print command.

Some commands can be entered with various parameters. For example,]PRINT 10 200 will print out the text in the text file with line numbers between 10 and 200. One must separate the mnemonic and the parameters from one another by at least one space. Do not use commas.

All disk filenames used by MAE are in the same format as described in the DOS manual. It is repeated as follows:

"Dx:name"

where: x is the disk drive number (default = 1)
and ´name´ is the 8 character filename and
3 character extender.

"D1:RABBIT.EXE"

A description of each text editor command follows:

]ASSEMBLE filename w

If filename is specified, load the disk file into text file and then begin assembly with contents of text file.

If w=LIST then generate a listing. If w=NOLIST or w not entered then an errors only output will be generated.

]AUTO x

Begin auto line numbering mode with next user entered line number. x specifies the increment to be added to each line number. You may exit auto line numbering by entering // immediately following the prompted line number.

]BREAK

Restore the zero page and go to the Monitor (MLM).

]CLEAR

Clear the text file.

]COPY x y z

Copy lines y thru z in the text file to just after line number x. The copied lines will all have line numbers equal x. At completion, there will be two copies of this data - one at x and the original at y. Since the copied lines are all the same line number,the text should be renumbered using the]NUMBER command.

]DC filename

This command will display the contents of the disk directory to the screen. Filename specifies the disk drive and filename to search for. For example,

]DC "D1:MEMTST.EXE"	Search drive 1 directory for the filename MEMTST.EXE. If file is found, it will be displayed along with the number of sectors remaining on the disk. If it is not found, only the number of sectors will be displayed.
]DC "D2:*.*"	Display all the files on drive 2 directory.
]DC	Display all the files on drive 1 directory. (If]DC is entered without without any parameters, all of the files on drive 1 directory will be displayed.

]DELETE x y

Delete entries in the text file between line numbers x and y. If only x is entered, only that line is deleted.

Note: Single lines can also be deleted by typing just the line number and pressing RETURN.

]EDIT t S1 t S2 t OR]EDIT n

String search and replace, or interline edit. See Part 7.

]FIND t S1 t

String search. See Part 7.

]FORMAT w n

Format the text file (where w=SET) or clear the format feature (where w=CLEAR). Format set tabulates the text file when outputted. This lines up the various source statement fields.

n specifies the number of characters per label (max. = 31). This is used to tabulate the listing.

]GET filename y

Get file from disk and store in the text buffer. If y is not entered, store at the start of the text buffer. If y is a line number, get the disk file and store it in the text buffer following the specified line number. If y = APPEND then get the file and store it after the last line number in the text file (that is, append it to the end of the current text).

A disk ´filename´ is specified in same format as shown in the

DOS manual and is repeated as follows:

"Dx:name"

where: x is the disk drive number (default = 1)
and 'name' is the 8 character filename and
3 character extender.

Examples are:
```
GET "D1:MAE.NOT"
GET "D2:RELOC.REL"
GET "D1:SUPPORT" APPEND
GET "D2:TEMP.INS" 1200
```

]HARD w x

Format for hard copy listing. This feature is designed to work with 66 line pages and leaves margin at top and bottom along with page number.]HA SET turns this feature on,]HA CLEAR turns this feature off. x is the starting page number.]HA PAGE advances to top of next page.

Each time]HA SET is entered, MAE resets its internal line counter to 0. Thus, you must manually adjust the paper in the printer (if you're using fan folded paper) so MAE and the printer are synced.

Note: This command only formats the page. It does not send it to the printer. To send this and other data to the printer, see the]TO command.

]LABELS w

Print out the entire contents of the label file if w=ALL or w not entered. Print only fixed (external) labels if w=FIXED. Print only internal or program labels if w=PROGRAM.

]MANUSCRIPT w

If w=SET, line numbers are not outputted when executing the]PR command. If w=CLEAR, line numbers are outputted when the]PR command is executed. Assembly output ignores the]MA command. If manuscript is to be generated using MAE, manuscript should be set and format clear (]MA SET,]FO CLEAR). Since the TED considers a blank line a deletion, you may insert a blank line by entering a line with a single period. When printed, a blank line will be output.

]MOVE x y z

Move lines y thru z in the text file to just after line number

x. The moved lines will all have line numbers equal to x. The original lines y thru z are deleted. Since the moved lines are all the same line number,the text should be renumbered using the]NUMBER command.

]NUMBER x y

Renumber the text file starting at line x in the text file and expanding by constant y. For example, to renumber the entire text file by 10, enter]NU 0 10.

]OUTPUT filename

Create a relocatable object file on disk. This command uses the 256 byte relocatable buffer that can be reallocated via the]SET command. Filename specifies the disk drive and filename to write to.

]PASS filename

Execute second pass of assembly. First pass must be previously performed. If filename is entered then the text file is loaded from disk before executing the second pass, else]PASS will assume the file is in the text file.

]PRINT x y

Print the text file data between line number x and y on the CRT. If only x is entered, only that line is printed. If no x and y, the entire file is printed.

Note: Use the control S, control Q, and control Z keys to control the output listing (see PART 4).

]PUT filename x y

Put text file between lines x and y to disk. If x and y are not entered, the entire text file will be put to disk.

A disk ´filename´ is specified in same format as shown in the DOS manual (see GET command).

]RUN label

Run (execute) a previously assembled program. If a symbolic label is entered, the label file is searched for the starting address. The called program should contain an RTS instruction as the last executable instruction.

]SET ts te ls le bs

If no parameters are given, the text file, label file, and relocatable buffer boundaries (addresses indicating text file start, end, label file start, end, and relocatable buffer start) will be output on the first line. On the second line the output consists of the present end of data in the text and label file. This command is commonly used to determine how much memory is remaining in the text file. If you are inputting hex digits for these addresses, preceed each with a '$' character.

If parameters are entered, the first two are text file start (ts) and end (te) addresses, then the label file start (ls) and end (le) addresses, and finally the relocatable buffer start address (bs).

Example :]SE $6800 $87FC $8800 $93FF $5800

Note : The relocating buffer is normally set at $5800 which is contained within MAE. Thus, unless desired, there is no reason to move the buffer.

]TO w

Assign terminal output to screen or printer. If w=PRINTER (or P), then output will be directed to both the screen and the printer. If w=ATARI (or A), terminal output is sent to only the screen.

Examples: TO PRINTER or TO P

TO ATARI or TO A

]USER

Restore zero page and go to location $0000. You must have entered a JMP instruction at that address.

B. ENTRY/DELETION/CHANGE OF TEXT

A source line of text is entered in the text file by entering a line number (0-9999) followed by the text to be entered. The line number string can be one to n digits in length. If the string is greater than 4 digits in length, only the right-most 4 are considered. Text may be entered in any order but will be inserted in the text file in numerical order. This provides for assembling, printing, and recording in numerical order. Any entry consisting of a line number with no text or just spaces results in a deletion of any entry in the text file with the same number. If text is entered and a corresponding line number already exists in the text file, the text with the corresponding number is deleted and the entered text is inserted.

To delete the entire file, use the]CL command.

To delete a range of lines, use the]DE command. To edit an existing line or lines having similar characteristics, use the]ED command.

To alter an existing line, use the]ED command form 2.

To find a string, use the]FI command. To move or copy lines use the]MO or]CO commands.

To insert a blank line, enter a line with just a period (.).

Text may be entered more easily by use of the auto line numbering feature (]AU command). Any]AU x where x does not equal 0 puts the TED in the auto line number mode on the next entry of a line number. Thus, the next line number will be automatically printed to the screen after the RETURN key is depressed. To exit from the automatic line numbering mode, type // following the current line number (Example-]3421//) and depress RETURN.

Note: While in the automatic line numbering mode, do not attempt to go back and change or correct a previously entered line. This will cause the current line number to have the wrong text associated with it. Corrections can be made after leaving the automatic mode.

When entering source for the assembler, one need not space over to line up the various fields. Labels are entered immediately after the line number. Separate each source field with one or more spaces. If the format feature is set (see]FO command), the TED will automatically line up the fields. Note: If a space is entered before the label, the TED will line up the label in the next field. This should result in an assembler error when assembled. Commands, mnemonics, and pseudo ops may be entered as upper case or lower case characters. Labels in the program may be entered as upper or lower case characters but a label entered as upper case will be unique to the same label entered as lower case.

5. ASSEMBLER (ASSM) FEATURES

The ASSM scans the source program in the text file. This requires at least 2 passes (or scans). On the first pass, the ASSM generates a label file (or symbol table) and outputs any errors that may occur. On the second pass, the ASSM creates an optional listing.

A third pass (via]OU), may be performed in order to generate a relocatable object file of the program in the text file. This file is recorded on disk and may be relocated at the users descretion practically anywhere in memory.

A. Source Statement Syntax

Each source statement consists of 5 fields as described below:

]line number	label	mnemonic	operand	comment

Label:

The first character of a label may be formed from the following characters:
@ A thru Z [\ ]

While the remaining characters which form the label may be constructed from the above characters and the following characters:
. / 0 thru 9 : ; < > ?

The label is always entered immediately after the line number.

Mnemonic (or Pseudo Op):

The mnemonic or pseudo op is separated from the label by one or more spaces and consists of a standard 6502 mnemonic of table A, pseudo op of table B, or macro name.

Operand:

The operand is separated from the mnemonic or pseudo op by one or more spaces and may consist of a label expression from table C and symbols which indicate the desired addressing mode from table D.

Comment:

The comment is separated from the operand field by one or more spaces and is free format. A comment field

begins one or more spaces past the mnemonic or pseudo op if the nature of such does not require an operand field. A free format comment field may be entered if a semicolon (;) immediately follows the line number.

NOTE: It is permissable to have a line with only a label. This is commonly done to assign two or more labels to the same address. If the line has only a label or label with comment, then the label may be any length up to 79 characters regardless of the label length set with the]FORMAT command.

TABLE A - 6502 Mnemonics

(For a description of each mnemonic, consult the 6502 Software Manual)

ADC	CLD	LDA	SBC
AND	CLI	LDX	SEC
ASL	CMP	LDY	SED
BCC	CPX	LSR	SEI
BCS	CPY	CLV	STA
BEQ	DEC	ORA	STX
BIT	DEX	PHA	STY
BMI	DEY	PHP	NOP
BNE	EOR	PLA	TAX
BPL	INC	PLP	TAY
BRK	INX	ROL	TSX
BVC	INY	ROR	TXA
BVS	JMP	RTI	TXS
CLC	JSR	RTS	TYA

Pseudo Ops are commands used internally by the assembler to cause it to perform certain functions or to tell it information to be used during the assembly process.

TABLE B - Pseudo Ops

.BA label exp.

Begin assembly at the address calculated from the label expression. This address must be defined on the first pass or an error will result and the assembly will halt.

.BY

Store bytes of data. Each hex, decimal, or binary byte must be separated by at least one space. An ascii string may be entered by beginning and ending with apostrophes (´). Example: .BY 00 ´ABCD´ 47 69 ´Z´ $FC %1101

.CE

Continue assembly if errors other than !07, !04, and !17 occur. All error messages will be printed.

.CT

Designate current contents of text buffer as a control file. Only one control file may exist during each assembly. Designation as a control file allows the use of .FI pseudo ops to link other files for the assembly process.

Note: Only one .EN pseudo op is allowed in each assembly and if .CT is used, the .EN must be at the end of that file. Thus, files referenced via .FI must not have a .EN pseudo op.

---

label .DE label exp.

Assign the address calculated from the label expression to the label. Designate as external and put in the label file. An error will result if the label is omitted.

---

label .DI label exp.

Assign the address calculated from the label expression to the label. Designate as internal and put in the label file. An error will result if the label is omitted.

.DS label exp.

Define a block of storage. For example, if label exp. equated to 4, then ASSM will skip over 4 bytes.

Note: The initial contents of the block of storage is undefined.

.EC

Suppress output of macro generated object code on source listing. This is the default state. See part 5E.

.EJ

Eject to top of next page if]HA SET was previously entered.

.EN

Indicates the end of the source program.

.ES

Output macro generated object code on source listing. See part 5E.

.FI filename

Assemble the specified file before continuing with statement following .FI.

Note: The .FI pseudo op is allowed only in the control file (that designated with .CT).

Note: The filename is in the same format as specified in the DOS manual (see GET command). For example -

```
.FI  "D1:COUNT.TMP"
```

.IN label

Output ? followed with space and then accept exactly 4 hex digits. These hex digits will be assigned to label and stored in the label file.

Input will only occur on the first pass of assembly. The label must be symbolic and should be defined similar to the following example:

```
1130              .PR "ENTER ASSEMBLY START"
1140BEGIN.ADDR    .IN BEGIN.ADDR
```

One should avoid using .DE, .DI, or SET to define the label as these constructs reassign their specified value on each pass.

.LC

Clear the list option so that the assembly terminates printing the source listing after the .LC on pass 2.

.LS

Set the list option so that the assembly begins printing out the source listing after the .LS on pass 2.

.MC label exp.

When storing object code, move code to the address calculated from the label expression but assemble in relation to that specified by the .BA pseudo op. An undefined address results in an immediate assembly halt.

.MD

Macro definition. See part 5E.

.ME

Macro end of definition. See part 5E.

.MG

.MG declares the entire contents of the text file as Macro Global. When assembling from disk, all following files will be loaded into the text file area following the file with the .MG. Thus, even though there can be many modules loaded and assembled, the macro global file is "locked" into the text file area providing its macro definitions for use by all subsequent files.

.OC

Clear the object store option so that object code after .OC is not stored in memory. This is the default option.

.OS

Set the object store option so that object code after the .OS is stored in memory on pass 2.

.PR "text"

Output the text that is enclosed in quotes when the .PR is encountered. MAE automatically issues a carriage return immediately before outputting the text. The text will be output only during the first pass of the assembly.

.RC

Provide directive to the relocating loader to stop resolving address information in the object code per relocation requirements and store code at the pre-relocated address. This condition remains in effect until a .RS pseudo op is encountered.

.RS

Provide directive to the relocating loader to resolve address information in the object code per relocation, and store the code at the proper relocated address. This is the default condition.

.SE label exp.

Store the address calculated from the label expression in the next two memory locations. Consider this address as being an external address. Note: If a label is assigned to the .SE, it will be considered as internal.

.SI label exp.

Store the address calculated from the label expression in the next two memory locations. Consider this address as being an internal address.

TABLE C - Label Expressions

A label expression must not consist of embedded spaces and is constructed from the following:

Symbolic Labels:

One to 31 characters consisting of the ASCII characters as previously defined. The maximum number of characters is set by the]FORMAT SET n command where n = the maximum number allowed. The default maximum is 10 characters per label.

Non-Symbolic Labels:

Decimal, hex, or binary values may be entered. If no special symbol preceeds the numerals then the ASSM assumes decimal (example: 147). If $ preceeds, then hex is assumed (example: $F3). If % preceeds, then binary is assumed (example: %11001). Leading zeros need not be entered. If the decimal or hex string is greater than 4 digits, only the rightmost 4 are considered. If the binary string is greater than 8, only the rightmost 8 are considered.

Program Counter:

To indicate the current location of the program counter, use the symbol =.

Arithmetic Operators:

These are used to separate the above label expression elements. Two operators are reconized:

+ addition
- subtraction

Examples of some valid label expressions follow:

```
LDA  #%1101              ;LOAD IMMEDIATE $0D
STA  *TEMP+$01           ;STORE AT BYTE FOLLOWING TEMP
LDA  $471E36             ;LOAD FROM LOCATION $1E36
JMP  LOOP+C-$461         ;JMP TO CALCULATED ADDRESS
BNE  =+8                 ;BRANCH TO CURRENT PC PLUS 8 BYTES
```

One special label expression is A, as in ASL A. The letter A followed with a space in the operand field indicates accumulator addressing mode. Thus LDA A is an error condition since this addressing mode is not valid for the LDA mnemonic.

ASL A+0 does not result in accumulator addressing but instead references a memory location.

TABLE D - Addressing Mode Formats

```
Immediate:
     LDA  #%1101        ;BINARY
     LDA  #$F3          ;HEX
     LDA  #MASK         ;SYMBOLIC
     LDA  #'A           ;ASCII
     LDA  #H,label exp.  ;HI PART OF THE ADDRESS OF THE LABEL
     LDA  #L,label exp.  ;LO PART OF THE ADDRESS OF THE LABEL

Absolute:
     LDA  label exp.

Zero Page:
     LDA  *label exp.     ;THE ASTERISK (*) INDICATES ZERO PAGE

Absolute indexed:
     LDA  label exp.,X
     LDA  label exp.,Y

Zero Page Indexed:
     LDA  *label exp.,X
     LDA  *label exp.,Y

Indexed Indirect:
     LDA  (label exp.,X)

Indirect Indexed:
     LDA  (label exp.),Y

Indirect:
     JMP  (label exp.)

Accumulator:
     ASL  A              ;LETTER A FOLLOWED WITH A SPACE INDICATES
                         ;ACCUMULATOR ADDRESSING MODE

Implied:
     TAX                 ;OPERAND FIELD IGNORED
     CLC

Relative:
     BEQ  label exp.
```

B. Label File (or Symbol Table)

A label file is constructed by the assembler and may be outputted at the end of assembly (if a .LC pseudo op was not encountered) or via the]LA command. The output consists of the symbolic label and its hex address. Via the]LA command, the user may select which type of labels to be output.]LA FIXED outputs all program and internal labels, and]LA ALL outputs all labels. When a relocatable object file is generated (via]OU command), any instruction which referenced an internal label or a label expression which consisted of at least one internal label will be tagged with special information within the relocatable object file. The relocating loader uses this information to determine if an address needs to be resolved when the program is moved to another part of memory.

Conversely, instructions which referenced an external label or a label expression consisting of all external references will not be altered by the relocating loader.

At the end of the label file the number of errors which occurred and program break in the assembly will be outputted in the following format: //xxxx,yyyy,zzzz

Where xxxx is the number of errors found in decimal representation, yyyy is last address in relation to .BA, and zzzz is last address in relation to .MC.

C. Assembling

Source for a large program may be divided into modules, entered into the text file one at a time and recorded (]PUT) on disk.

These modules can be linked together during assembly via a ´control file´. If used, the control file must be the first file to be assembled. This file must be in the text buffer when the]AS command is issued, or its name must be specified in the]AS command (example:]AS "D1:MEM.CTL"). Files are linked together via the .FI pseudo op. For example, to assemble 3 files named X.M01, Y.M02, and Z.M03, we need to generate a control file say M.CTL (note for convenience we use the convention of tagging CTL on the end of any name which references a control file while its modules are tagged Mxx). The file M.CTL may contain the following:

```
.CT
.FI   "D1:X.M01"   ;FIRST DISK FILE TO BE ASSEMBLED
.FI   "D1:Y.M02"   ;SECOND DISK FILE
.FI   "D1:Z.M03"   ;THIRD DISK FILE
.EN
```

Now, when the control file is assembled, MAE is told to go assemble the files in the order specified.

At assembly, the assembler can load and assemble each module until the entire program has been assembled. This will require two pass for a complete assembly. When the end of a pass is encountered, MAE will output the message END MAE PASS!. If for some reason you terminate the assembly on the second pass, you may restart at the beginning of the second pass using the]PASS command.

D. Creating a relocatable object file (]OU)

In order to create a relocatable object file, the programmer should identify those labels whose addresses are fixed and should not be altered by the relocating loader. This is done via the .DE pseudo op. Non-symbolic labels (example: $0169) are also considered as being external (or fixed). All other labels (including those defined via the .DI pseudo op) are considered as internal. Addresses associated with internal labels can be altered by an offset when the program is loaded via the relocating loader.

Also, the .SE stores a two byte external address and the .SI stores a two byte internal address. Similarily the relocating loader will alter the internal address and not the external address.

An example of an external address would be the calls to ATARI ROM routines or any location whose address remains the same no matter where the program is located. Expressions consisting of internal and external labels will be combined and considered an internal address. A label expression consisting entirely of external labels will be combined and considered as external.

The relocating loader can relocate your program in 3 segments: Zero page variables (internal addresses in range $00-$FF), absolute variables (internal addresses in range $0400-$1FFF), and program body (references in range $2000-$FFFF). To generate a relocatable object file, first partition your program into internal and external references. Remember, external references are those locations that are fixed while internal references are those locations which can be altered by the relocating loader.

Start assigning zero page references at location $0000, absolute variable locations at $0400, and begin assembly of the program at $2000. Next assemble the program via]AS, and then issue the]OUT command to generate a relocatable object file.

Now, we have the relocatable object code on disk. To load this object code back into memory, first load the relocating loader. The

relocating loader is contained on the diskette with the name RELOC.EXE . To load the relocating loader, exit the MAE and the MLM and return to the ATARI DOS menu. Use the 'load' (L) command to load in the RELOC.EXE. Execution begins at $2683 if in the MLM or with RUN $2683 if in the MAE. The relocating loader will request the following:

1) FILENAME? Name of the file containing the relocatable object code.

2) Z-PG OFFSET? Address to begin assignment of zero page internal references.

3) ABS OFFSET? Address to begin assignment of absolute internal references.

4) PGM EXE OFFSET? Address the program is to execute.

5) PGM STORE OFFSET? Address to store the program object code.

When the file has been relocated in memory, it can be saved on disk (using the DOS) as an executable file, which may be reloaded without using the relocating loader.

When the relocating loader has finished its work, it will come back and print FILENAME?. To exit the program, simply depress RETURN and the relocating loader will break to the MLM.

As an example, lets assume we want to relocate a program named UART to execute at location $3000, but store the object code at $8000, and start the zero page variables at $0080, and the absolute variables at $7000. We would respond to the relocating loader as follows:

```
FILENAME?  "D1:UART.REL"          _ File name
---------
Z-PG OFFSET?  80                  _ Assign start of zero page var.
------------
ABS OFFSET?  7000                 _ Assign start of absolute var.
-----------
PGM EXE OFFSET?  3000             _ Program body start
---------------
PGM STORE OFFSET?  8000           _ Store of code start
-----------------
```

```
LOAD MAP
--------
   :                    _ Relocating loader
   :                    _ outputs a load map
   :

FILENAME?               _ Enter just RETURN to exit the
---------                 Relocating Loader
```

E. Macros

MAE provides a macro capability. A macro is essentially a facility in which one line of source code can represent a function consisting of many instruction sequences. For example, the 6502 instruction set does not have an instruction to increment a double byte memory location. A macro could be written to perform this operation and represented as INCD (VALUE.1). This macro would appear in your assembly language listing in the mnemonic field similar to the following:

```
1000  BNE  SKIP
1010  NOP
        .
        .
1060  INCD (VALUE.1)    ;MACRO -INCREMENT DOUBLE
1070  LDA  TEMP
        .
        .
```

Before a macro can be used, it must be defined in order for ASSM to process it. A macro is defined via the .MD (macro definition) pseudo op. Its form is :

```
!!!label   .MD  (L1  L2  ...  Ln)
```

Where label is the name of the macro (!!! must preceed the label), and L1, L2, ..., Ln are dummy variables used for replacement with the expansion variables. These variables should be separated using spaces, do not use commas.

To terminate the definition of a macro, use the .ME (macro end pseudo op).

For example, the definition of the INCD (increment double byte) macro could be as follows:

```
5430!!!INCD   .MD   (LOC)     ;INCREMENT DOUBLE
5440          INC   LOC
5450          BNE   SKIP
5460          INC   LOC+1
5470SKIP      .ME
```

This is a possible definition for the macro INCD. The assembler will not produce object code for the MACRO until there is a call for expansion (see .ES pesudo op).

Note: A call for expansion occurs when you enter the macro name along with its parameters in the mnemonic field as INCD (TEMP) or INCD (COUNT) or INCD (COUNT+2) or any other labels or expressions you may choose.

Note: In the expansion of INCD, code is not being generated which increments the variable LOC but instead code for the associated variable in the call for expansion.

If you tried to expand INCD as described above more than once, you will get a !06 error message. This is a duplicate label error and it would result because of the label SKIP occurring in the first expansion and again in the second expansion.

There is a way to get arround this and it has to do with making the label SKIP appear unique with each expansion. This is accomplished by rewriting the INCD macro as follows:

```
5430!!!INCD   .MD (LOC)       ;INCREMENT DOUBLE
5440          INC  LOC
5450          BNE  ...SKIP
5460          INC  LOC+1
5470...SKIP   .ME
```

The only difference is ...SKIP is substituted for SKIP. What the ASSM does is to assign each macro expansion a unique macro sequence number (2**16 maximum macros in each file). If the label begins with ... then ASSM will assign the macro sequence number to the label. Thus, since each expansion of this macro gets a unique sequence number, the labels will be unique and the !06 error will not occur.

If the label ...SKIP also occurred in another macro definition, no !06 error will occur in its expansion if they are not nested. If you nest macros (i.e. one macro expands another), you may get a !06 error if each definition uses the ...SKIP label. The reason this may occur is that as one macro expands another in a nest, they each get sequentially assigned macro sequence numbers. As the macros work out of the nest, the macro sequence numbers are decremented until the top of the nest. Then as futher macros are expanded, the sequence numbers are again incremented. The end result is that it is possible for a nested macro to have the same sequence number as one not nested

or one at a different level in another nest. Therefore, if you nest macros, it is suggested that you use different labels in each macro definition.

Some futher notes on macros are:

1) The macro definition must occur before the expansion.

2) The macro definition must occur in each file that references it. Each file is assigned a unique file sequence number (2**16 maximum files in each assembly) which is assigned to each macro name. Thus the same macro can appear in more than one file without causing a !06 error. If a macro with the same name is defined twice in the same file, then the !06 error will occur.

3) Macros may be nested up to 32 levels. This is a limitation because there is only so much memory left for use in the stack.

4) If a macro has more than one parameter, the parameters should be separated using spaces - do not use commas.

5) The number of dummy parameters in the macro definition must match exactly the number of parameters in the call for expansion.

6) The dummy parameters in the macro definition must be symbolic labels. The parameters in the expansion may be symbolic or non-symbolic label expressions.

7) If the .ES pseudo op is entered, object code generated by the macro expansion will be output in the source listing. Also, comment lines within the macro definition will be output as blank lines during expansion. Conversely, if .EC was entered, only the line which contained the macro call will be output in the source listing.

8) A macro name may not be the same as a 6502 mnemonic, pseudo op, or conditional assembly operator.

F. Conditional Assembly

MAE also provides a conditional assembly facility to conditionally direct the assembler to assemble certain portions of your program and not other portions. For example, assume you have written a CRT controller program which can provide either a 40, 64, or 80 character per line display. Instead of having to keep 3

different copies of the program, you could use the ASSM conditional assembly feature to assemble code concerned with one of the character densities.

Before we continue with this example, lets describe the Conditional Assembly operators:

IFE label exp.

If the label expression equates to a zero quantity, then assemble to end of control block.

IFN label exp.

If the label expression equates to a quantity not equal to zero, then assemble to end of control block.

IFP label exp.

If the label expression equates to a positive quantity or 0000, then assemble to end of control block.

IFM label exp.

If the label expression equates to a negative (minus) quantity, then assembly to end of control block.

***

Three asterisks in the mnemonic field indicates the end of the control block.

SET label=label exp.

Set the previously defined label to the quantity calculated from the label expression.

NOTE: All label expressions are equated using 16 - bit precision arithmetic.

Going back to the CRT controller software example, a possible arrangement of the program is as follows:

```
CHAR.LINE     .DE  40
                .
                .
              IFE  CHAR.LINE-40
;CODE FOLLOWS FOR 40 CHARACTER PER LINE
                .
                .
              ***

              IFE  CHAR.LINE-64
;CODE FOLLOWS FOR 64 CHARACTER PER LINE
                .
                .
              ***

              IFE  CHAR.LINE-80
;CODE FOLLOWS FOR 80 CHARACTER PER LINE
                .
                .
              ***

;COMMON CODE FOR ALL
                .
                .
```

Shown is the arrangement which would assemble code associated with 40 characters per line since CHAR.LINE is defined as equal 40. If you wanted to assemble for 80 characters, simply define CHAR.LINE as equal 80.

Conditional assembly can also be incorporated within macro definitions. A very powerful use is within a macro you don't want it completely expanded each time it is referenced. For example, assume you wrote a macro to do a sort on some data. It could be defined as follows:

```
EXPAND    .DE  0
!!!SORT   .MD
          IFN  EXPAND
          JSR SORT.CALL     ;CALL SORT
          ***

          IFE  EXPAND
          JSR  SORT.CALL
          JMP  ...ABC

;SORT CODE FOLLOWS
SORT.CALL
```

```
          .
          .
         RTS

...ABC   SET  EXPAND=1
         ***

         .ME
```

In this example, EXPAND is initially set to 0. When the macro is expanded for the first time, EXPAND equals 0 and the code at SORT.CALL will be assembled along with a JSR to and a JMP around the sort subroutine. Also, the first expansion sets EXPAND to 1. On each suceeding expansion, only a JSR instruction will be assembled since EXPAND equals 1. Using conditional assembly in this example resulted in more efficient memory utilization over an equivalent macro expansion without conditional assembly.

G. Interactive Assembly

Interactive assembly is a new concept in which the assembler can be instructed to print messages and/or accept keyboard input during the first pass of the assembly.

Interactive assembly makes use of two pseudo ops:

```
.PR  to print messages
.IN  to accept keyboard input
```

An example of the use of interactive assembly is as follows:

```
         .PR  "INPUT START OF ASSEMBLY"
ADDR
         .IN  ADDR
         .BA  ADDR
```

Note that in this example, the assembler will request entry of an address to be assigned to ADDR, and then begins assembly at that address.

There are many applications for interactive assembly but those possibilities are left for the users of MAE.

NOTE: Never specify a label as the operand in the .IN pseudo op that has been defined by the .DE, .DI, or SET pseudo ops. The reason is that these pseudo ops initialize the address assigned to associated labels on both assembly passes while all other labels are initialized only on the first pass. Since the .IN pseudo op accepts input on the

first pass only, usage of labels defined by .DE, .DI, and SET will cause different label values on pass 1 versus pass 2.

H. Default Parameters on entry to ASSM

- Does not store object code in memory (otherwise use .OS)
- Begins assembly at $0400 (otherwise use .BA)
- Halts assembly on errors (otherwise use .CE)
- Stores object code beginning at $0400 unless a .BA or .MC is encountered and if .OS is present.
- Object code generated by macros does not appear on the assembly listing (i.e. default is .EC)

6. ERROR CODES

An error message of the form !xx AT LINE yyyy where xx is the error code and yyyy is the line number will be outputted if an error occurs. Sometimes an error message will output an invalid line number. This occurs when the error is on a non-existant line such as an illegal command input.

The following is a list of error codes not specifically related to macros:

ERROR CODE	DESCRIPTION
1B	.EN in non .CT file when .CT file exists.
1A	.EN missing in .CT designated file.
19	Found .FI in non .CT file.
18	
17	Checksum error on disk load.
16	
15	Syntax error in]ED command.
14	Device numbers 0,1,2,3 not allowed.
13	Multiple .CT assignment.
12	Command syntax error or out of range error.
11	Missing parameter in]NU command.
10	Overflow in line # renumbering. CAUTION: You should properly renumber the the text file for proper command operation.
0F	Overflow in text file - line not inserted.
0E	Overflow in label file - label not inserted.
0D	MAE expected hex characters, found none.
0C	Illegal character in label.
0B	Unimplemented addressing mode.
0A	Error in or no operand.
09	Found illegal character in decimal string.

08	Undefined label (may be illegal label).
07	.EN pseudo op missing.
06	Duplicate label.
05	Label missing in .DE or .DI pseudo op.
04	.BA or .MC operand undefined.
03	Illegal pseudo op.
02	Illegal mnemonic or undefined macro.
01	Branch out of range.
00	Not a zero page address.
ED	Error in command input.

The following is a list of error codes that are specifically related to macros and condition assembly:

ERROR CODE	DESCRIPTION
2F	Overflow in file sequence count (2**16 max.)
2E	Overflow in number of macros (2**16 max.)
2D	
2C	
2B	.ME without associated .MD
2A	Non-symbolic label in SET pseudo op.
29	Illegal nested definition.
28	
27	Macro definition overlaps file boundary.
26	Duplicate macro definition.
25	Quantity parms mismatch or illegal characters.
24	Too many nested macros (32 max.)
23	Macro definition not complete at .EN
22	Conditional suppress set at .EN
21	Macro in expand state at .EN
20	Attempted expansion before definition.

In addition to the MAE error codes discussed above, this software will also gives ATARI system errors in the form SYSTEM ERROR=xxxx -where xxxx is one of the error code messages shown in the BASIC and DOS manuals. These error messages indicate an error was given by the ATARI operating operating system to MAE. For example, a SYSTEM ERROR=170 says a disk file was not found.

7. STRING SEARCH AND REPLACE COMMANDS

A. Edit Command

A powerful string search and replace, and line edit capability is provided via the]EDIT command to easily make changes in the text file. Use form 1 to string search and replace, and form 2 to edit a particular line.

Form 1

```
]EDIT  tS1tS2t  %d  *  x  y
                    #
```

Where:
- t is a non-numeric, non-space terminator
- S1 is the string to search for
- S2 is the string to replace S1
- d is don't care character. Preceed with % character to change the don't care, else don't care character will be % by default.
- * indicates to interact with user via subcommands before replacing S1
- # indicates to alter but provide no printout
  Note: No * or # indicates to alter and provide printout.
- x line number start in text file
- y line number end in text file

Asterisk (*) prompter subcommands:

A	alter field accordingly
D	delete entire line
M	move to next field - don't alter current
S	skip line - don't alter
X	exit]ED command
2	enter form 2

Defaults:
- d = %
- x = 0
- y = 9999

If no * or # entered then print all lines altered.

For example, to replace all occurances of the label LOOP with the label START between lines 100 and 600, enter:

```
]EDIT  /LOOP/START/  100  600
```

To simply delete all occurances of LOOP, enter:

```
]EDIT  /LOOP//  100  600
```

You may use the * and # as described above.

The slash ("/") was used in the above examples as the terminator but any non-numeric character may be used.

At the end of the]EDIT operation, the number of occurances of the string will be output as //xxxx where xxxx is a decimal quantity.

Form 2

]EDIT n

Where: n is the line number (0-9999) of the line to be edited.

After executing the command, cursor over to the part to be changed, and either type over or use the INSERT/DELETE key on the ATARI as you would use the screen editor. Press RETURN when done, and MAE will insert it in the text file.

B. Find Command

If you want to just find certain occurances of a particular string, use the]FIND command. Its form is:

]FIND tSlt # x y

Where: t, Sl, #, x, and y are as defined in EDIT command.

For example,]FIND /LDA/ will output all occurances of the string LDA in the text file.

At the end of the]FIND operation, the number of occurances of the string will be output as //xxxx where xxxx is a decimal quantity.

A unique use of this command is to count the number of characters in the text file (excluding line numbers). The form for this is:]FIND /%/#

8. EXAMPLES

A. TED Examples

#1 Illustrate entry of text.

```
]AUTO  10
]1000;THIS IS A TEST
 1010LOOP LDA VALUE,Y
 1020  NOP
 1030END.PGM  .EN
 1040//              _Note, enter // to exit
                           auto line numbering
```

#2 Illustrate listing of text.

```
]PRINT
 1000 ;THIS IS A TEST
 1010 LOOP       LDA VALUE,Y
 1020            NOP
 1030 END.PGM    .EN
//
```

#3 Put file to disk drive number 1 with name TEST.

```
]PUT  "D1:TEST"
```

#4 Get file from disk drive number 1 named TEST.

```
]GET  "D1:TEST"
```

#5 Assemble file CRTDVR from drive 1 and generate a listing.

```
]ASSM  "D1:CRTDVR"  LIST
```

#6 Find all occurrances of the text LDA.

```
]FIND  /LDA/
```

#7 Replace all occurances of LDA FA with LDA *FA between lines 1000 and 2000.

```
]EDIT  /LDA FA/LDA *FA/ 1000 2000
```

#8 Provide for 15 caracters per label.

```
]FORMAT  SET  15
```

#9 Output all fixed (external) labels.

```
]LABELS FIXED
```

#10 Renumber the text file beginning at line number 100 and incrementing by 5.

```
]NUMBER 100 5
```

#11 Move lines 100 thru 200 to after line 9000

```
]MOVE 9000 100 200
```

#12 Print lines 900 thru 976

```
]PRINT 900 976
```

#13 Reallocate the text file to $6800 thru $BC1C

```
]SET $6800 $BC1C
```

#14 Go to Machine Language Monitor.

```
]BREAK
```

#15 Run assembly program at symbolic label BOX.

```
]RUN BOX
```

B. ASSM Examples

#1 Begin assembly at $600 and store object code.

```
.BA  $600
.OS
```

#2 Begin assembly at $0700 but store object code at $8000.

```
.BA  $0700
.MC  $8000
.OS
```

#3 Define the CRT output routine.

```
CRT   .DE  $F6A4
```

#4 Assign an internal work location in zero page.

```
WORK  .DI  $0
```

#5 Allocate 6 bytes of storage.

```
TABLE  .DS  6
```

#6 Define label EOI as mask with bit 6 set and show use in AND statement.

```
EOI      .DE  %01000000
         AND  #EOI
```

#7 Load the low address part of the label VALUES in register X and high part in register Y.

```
LDX  #L,VALUES
LDY  #H,VALUES
```

#8 Give example of .BY pseudo op.

```
.BY  'ALARM CONDITION ON MOTOR 1'  $9B
```

#9 Store the address of the internal label TABLE and the external label ATROUT.

```
.SI  TABLE
.SE  ATROUT
```

#10 Define the contents of the text.file as Macro Global so its macro definitions can be used by subsequent files in the assembly.

```
.MG
```

NOTE: This locks the macro definitions in the text buffer. If you get a !0F error on subsequent loads, you should know that you have overflowed the text buffer. The solution is to allocate more memory (via]SET command) and then reassemble.

#11 Show example of a very long label.

```
MEMORY.TEST.FOR.6502
            JMP MEMORY.TEST.FOR.6502
```

NOTE: Long labels (greater than that specified via]F0 command) are allowed if defined on a line with no mnemonics.

9. GETTING STARTED WITH MAE

The supplied diskette contains its own disk operating system (DOS), MLM, and MAE files. Except for removing the BASIC cartridge, there are no other special instructions for loading the files. Just insert the diskette and boot the system in the normal way. All files will be automatically loaded when the DOS is loaded.

When the DOS boot process is completed, the DOS menu is displayed. Notice the ´0´ command on the menu. Simply type 0 and press RETURN to enter the MLM. As discussed in Part 3, the MLM provides the ability to interact with 6502 and memory. Now to leave the MLM and enter the MAE, type AC and press RETURN. (This is the cold start entry command which is built in to the MLM.)

MAE will respond with:

```
C 1981 by EHS

6800-7C1C   2680-2FFC   5800
6800   2680

]
```

This displays the default allocations of memory for the text file (6800-7C1C), label file (2680-2FFC), and start address of the 256 byte relocatable buffer (5800). On the next line, the current end of the text file and label file are displayed. Since they are initially cleared, these are the same as their respective start addresses. You should note that the current end will change as you insert/delete data in the text file and label file. The]SET command can be used to display this range again or alter the file boundaries.

Note: On any entry or exit from MAE, MAE will swap the zero page area that it uses ($80 to $FF) with a safe area at $5680. Therefore a users program can make use of $80 to $EF without affecting the MAE variables.

Remember, to exit MAE, issue the]BREAK commands to return to the MLM. You may reenter MAE from the MLM by typing AW command (warm start - everything preserved) or AC command (cold start - everything cleared to default state).

The first thing you should do now is to load the MAE1.NOT file via:

```
]GET "D1:MAE1.NOT"
]FORMAT CLEAR       ; turn formatting off
]PRINT              ; print the file
```

The MAE1.NOT file will contain any pertinent information pertaining to MAE that was discovered after this manual was printed. Please review the information in this file.

Now you should start playing around with MAE by executing its commands and then proceeding to entering programs. Try reviewing the commands in PART 4, assembler features in PART 5, and then the examples in PART 8.

In addition, we have also included several MAE source files on the diskette for your use in understanding how MAE functions. Use the]DC command to display the disk files. Notice all the files listed as EXAMPxx ASM where xx is a number. All of the EXAMP files contain examples MAE source files. For example, type]GET "D:EXAMP01.ASM". When the file is loaded, type]AS L to assemble the file and display it to the screen.

We hope you find MAE to be an excellent program development aid and a worthwhile investment. Happy Assembling!!!

10. MAE Simplified Text Processor (STP)

The MAE Simplified Text Processor (STP) is a word processor program designed specifically to work with the MAE text editor. The primary purpose of this word processor was to provide a simplified means to process program documentation and for other text processing needs. This simplicity was accomplished with a set of 21 easily remembered word processing functions, and usage of an already familiar text editor to enter and edit the raw text.

To instruct the word processor to perform a word processing function, one inserts text macros in the text to be formatted. A text macro always begins with a period (.), always begins in column

1, may be entered as upper or lower case, and may or may not have associated parameters. The following are the macros provided by the STP word processor:

VERTICAL SPACING (.vspace n)

This macro is used to provide single, double, triple spacing, etc. for the entire output. Enter the macro as shown above with the desired spacing. For example, to request a double spaced output, enter .vspace 2.

TEMPORARY INDENT (.sn)

To indent n spaces on the next line, use the .sn macro where n = the number of spaces to indent. For example, .s5 will indent the next line 5 spaces from the right.

MARGIN CONTROL (.m n p q r)

The margins default to 66 lines per page, left margin begins at column 0, print width = 76 characters per line, and the number of blank lines between text body and each title and footer = 3.

The parameters in the margin macro are:

- n = left margin begin position (default = 0)
- p = number of characters per line (default = 76)
- q = number of lines per page minus r. Example if lines per page = 66 and the number of blank lines between titles and footers = 3, then q = 66-3 = 63.
- r = number of blank lines between text body and each header and footer. Default = 3.

For example to specify left margin to begin in column 5, print width of 60, 66 lines/page, and 4 spaces between text body and titles and footers, enter .m 5 60 62 4.

If you enter just .m 5 60, the previously entered values for parameters q and r will be assumed. The margin may be changed at any point as desired in the text. The maximum value for n is 76.

TURN OFF JUSTIFICATION (.nofill)

The .nofill macro turns off the justification function. That is, all lines will be outputted exactly as typed and formatted. This means that the lines will be printed without adding spaces to make

the margins come out even. Also, words are not combined to fill to the specified margins. (See .ju command.)

BEGIN A NEW PAGE (.ff)

The .ff macro may be entered when one wants the printer to eject to the top of the next page.

LITERAL SPACE (^character)

Normally, spaces are not processed like other characters. If several spaces are entered consecutively, the STP word processor reconizes only one space and deletes the rest. If it is desired to force a certain number of spaces in a line for tabular formats, etc., a string of caret (^) characters may be inserted into the text. The caret will not be printed when the text is processed but instead a space will be printed for

It is also possible to change the literal space character by using a . x command -where x is the new literal space character. For example, a . # would change the literal space charater to a number sign (#). To have no character as a literal space, just enter . and return.

TURN ON JUSTIFICATION (.ju)

The .ju macro may be entered in order to restore justification. using the .nofill macro.

RAGGED RIGHT MARGIN (.rr)

This macro turns off the addition of spaces in order to make the margins come out even. Words are still combined in order to approximate the specified number of characters per line. The left margin will be straight but the right margin will be ragged.

RAGGED LEFT MARGIN (.rl)

This macro is the same as the .rr macro except that the right margin is straight and the left margin is ragged.

SKIP NEXT N LINES (.ln)

Use this macro to skip a number of lines before printing the next line of text. For example, to skip 2 lines and begin printing, enter .12. If you enter .1 by itself, one will be assumed. Thus .1 and .11 are equivalent and each will result in a movement to the next

line.

CENTER LINE OF TEXT (.c text)

This macro is useful for centering a line of text. For example, to center the phrase STP Word Processor, enter .c STP Word Processor.

SWAP JUSTIFICATION MODES (.swap)

This macro is used to switch from .rr mode to .rl and vice versa.

PARAGRAPH SPECIFICATION (.p d r) and PARAGRAPH IDENTIFICATION (.p)

Use the .p d r macro to inform the word processor what a paragraph is supposed to be: d = number of lines down, and r = number of spaces right for paragraph indent. The default is d = 1, and r = 5.

In order to identify a paragraph start in your text, use the .p macro with no parameters.

PAGE TITLE (.t# title text)

A one line title at the top of each page may be entered using this macro. For example, to specify the title CONFIDENTIAL, enter .t CONFIDENTIAL. If you want to also include a page number, enter .t# CONFIDENTIAL. Note that the # specifies page numbering. If you want just a page number (the default state), enter just .t#. If you want neither title nor page number, enter just .t to turn off all titling.

PAGE FOOTERS (.foot# foot text)

A one line footer at the bottom of each page may be specified using this macro. The parameters for .foot are the same as for the .title. The default is no footers.

MONOSPACED PRINTING (.mono)

The ATARI 825 printer has the capability of printing different character sets. One of these is 10-cpi monospaced. By using the .mono command, the STP will cause the printer to select this character set.

CONDENSED PRINTING (.cond)

The ATARI 825 printer has the capability of printing different character sets. One of these is 16.7-cpi condensed printing. By using the .cond command, the STP will cause the printer to select this character set.

PROPORTIONAL PRINTING (.prop)

The ATARI 825 printer has the capability of printing different character sets. One of these is proportional printing. By using the .prop command, the STP will cause the printer to select this character set.

CREATING SHAPE TABLES (.shape n and .set n 1 p)

The STP Word Processor has provisions for printing text in various shape formats by using a table to control the right and left margins. The .shape macro is used to define the shape to be used. Shape 1 is in the form of an 'I' and entered by simply entering the command .shape 1 at the beginning of the text file.

The .shape 2 macro may be used to create a user defined shape. In order to define the desired shape, .set macros are used to make entries in the user shape table corresponding to the desired shape. The parameters in the .set n 1 p are as follows:

n = line number for this margin specification
1 = column for left margin start
p = number of characters to be printed on this line

For example, .set 14 5 40 defines line 14 as left margin starts in column 5, and there are 40 characters to be printed on this line.

Normally one would have to enter 66 set macros to complete the user shape table. But it should be noted that .set 0 1 p is a special case. The 0 (which would normally represent the line number) indicates that all lines in the file are set to a left margin of 1 and print width of p. This is useful as you can set all lines in the user shape table to a particular margin and then use non 0 values to change certain lines to form the desired shape.

Note: Always enter the .shape 2 macro before the .set macros. The reason is that as soon as the .shape 2 macro is encountered, it fills the user shape table to default values of left margin = 0, and print width = 40. Thus if you enter .set macros first, they will be overwritten by the .shape 2 defaults of 0 and 40.

If .shape 2 is entered and no shape commands are entered, the

margins will default to .m 0 40. This is very useful when it is desired to view the formatted output on ATARI's which have 40 column screens.

LINK DISK FILES (.link filename)

This macro is used to link disk file modules together so they look like one large file. Thus, this feature allows text in multiple disk files to be processed by the STP. Filename is in the format as discussed in the DOS manual (see GET command).

Simply enter the .link macro at the bottom of the file immediately before the one you want loaded and formatted. Thus the first file will have a .LINK to the second, the second to third, etc. There is no limit to the number of files linked.

An example is: .LINK "D1:PART.M02"

DEFAULT CONDITIONS

The following are a number of assumed defaults that exist on initial entry to the word processor.

Justification = on
Shaping = off
Margins = 66 lines/page, 3 blank lines between text body and titles and footers, left margin = 0, and print width = 76.
Vertical Spacing = 1 (single spaced output)
Paragraph = 1 line down and 5 space indent
Page Title = page number but no text
Page Footer = no text or page number

ATARI PRINTER CONTROL CODES

If you have an ATARI 825 printer, it is possible to to use some of its PRINTER CONTROL CODES (see printer manual). The following is a list of control codes which can be entered into the word processing text.

CTRL-O	Start Underlining
CTRL-N	Stop Underlining
ESC ESC CTRL-N	Start Elongated Printing
ESC ESC CTRL-O	Stop Elongated Printing

Note-Once the ESC ESC character has been entered into the word processor text, it will not be displayed when printed to the screen. In addition, the ESC ESC character must be reentered if any changes are made to the line of text.

HOW TO USE THE STP WORD PROCESSOR

The STP word processor object code is stored on disk and must be loaded into memory using the DOS. The STP will be loaded into the memory space normally used by the label file, that is $2680 to $2FFF. Since the label file is not used in this word processing application, no conflict of memory space will occur. The following procedure can be used to load and run the STP.

1) Load the word processor using the DOS menu 'binary load' command. The file name is "D1:WORDP.EXE".

2) Enter the MLM using the O command on the DOS menu.

3) Type AC to enter MAE.

4) Type]FORMAT CLEAR to clear format mode.

5) Type RUN $2689 to initialize the STP. The screen will clear and display the file boundries. This initialization causes two new commands to be added to the TED command table. The commands are:

   ]WC L n and]WP L n

   The]WC command tells the word processor to format the text and output it only to the screen.

   The]WP command tells the word processor to format the text and output it to the screen and the printer.
   (Be sure the printer and interface are properly turned on.)

   The 'L' in the]WC and]WP commands is the optional link indicator. It enables the .link text macro. If 'L' is not entered, any .link command is skipped (see .link text macro).

   The 'n' in the]WC and]WP commands is an optional number which indicates page suspression. That is, if a number is entered, the word processor will format the text but suppress outputting that number of pages to the screen or printer. If if a number is not entered, then all pages will be outputted.

   Examples -]WC
   Send text to screen.

   ]WC L
   Send text to screen and link disk files.

   ]WC L 5
   Send text to screen,link disk files, and

skip first five pages.

]WP
Send to screen and printer.

]WP L
Send text to screen and printer and link disk files.

]WP L 5
Send text to screen and printer ,link disk files and skip the first five pages.

5) Enter raw text using MAE for editing. Include all necessary text processing macros.

6) When you are finished entering the raw text and associated text macros, use the]WC or]WP commands to output the formated text.

DISK FILE EXAMPLE

A raw text file named WORDP1.INS and WORDP2.INS is contained on the diskette. Type]GET "D1:WORDP1.INS" to load this file. Type]PRINT to examine the raw text with associated macros. Then type]WC L or]WP L to output the file.

Now compare the raw text printout with its text macros to the formatted output generated by the word processor. Examine these two printouts until you are familiar with the function of the STP macros.

11. SPECIAL NOTES

* When entering source modules (without .EN), you can perform a short test on the module by assembling the module while in the text file and watching for the !07 error. If other error messages occur, you have errors in the module. This short test is not a complete test but does check to insure you have lined up the fields properly, not entered duplicate labels within the module, or entered illegal mnemonics or addressing modes.

* An 80 character/line output device should be used when printing an assembly listing in order to provide a neat printout without foldover to the next line.

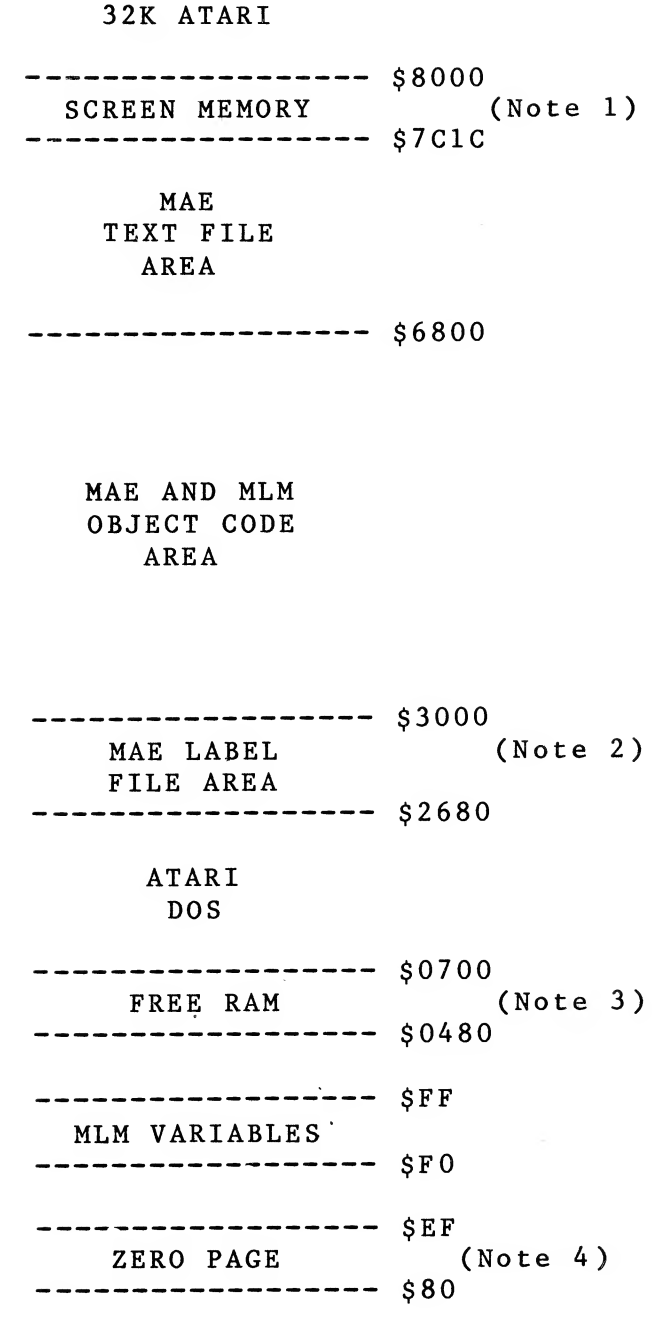
* We recommend that a naming convention for your files be established. We use the following extensions:

name.CTL	-	Control File
name.Mxx	-	Module referenced in Control File
name.ASM	-	Source file without .CT
name.EXE	-	Executable object file
name.REL	-	Relocatable object file
name.MAC	-	File containing all Macros
name.LIB	-	Library of symbols
name.LIM	-	Library of Macros
name.DOC	-	Program Documentation
name.INS	-	User instructions
name.NOT	-	Program Notes
name.BAS	-	Basic Program
name.DAT	-	Basic Data File

* Be carefull when using the]PUT command. Make sure the correct filename is used; otherwise, the wrong file may be written over and lost forever.

12. Memory Map

Shown below ia a memory map of ATARI memory and how it is used by this software.



Note 1: For 40K system, $9C1C is the highest usable RAM location. For 48K system, $BC1C is the highest. Care must be taken not to overwrite screen memory.

Note 2: This area is also used to store the object code for the STP word processor and the relocating loader software.

Note 3: The STP word processor uses this area for variable storage.

Note 4: MAE uses this zero page area. However before doing so, it copies the contents to a safe area at $5680. Therefore, a users program can use this zero page area without writing over MAE variables.

ERROR CODE	DESCRIPTION
1B	.EN in non .CT file when .CT file exists.
1A	.EN missing in .CT designated file.
19	Found .FI in non .CT file.
18	
17	Checksum error on disk load.
16	
15	Syntax error in]ED command.
14	Device numbers 0,1,2,3 not allowed.
13	Multiple .CT assignment.
12	Command syntax error or out of range error.
11	Missing parameter in]NU command.
10	Overflow in line # renumbering. CAUTION: You should properly renumber the the text file for proper command operation.
0F	Overflow in text file - line not inserted.
0E	Overflow in label file - label not inserted.
0D	MAE expected hex characters, found none.
0C	Illegal character in label.
0B	Unimplemented addressing mode.
0A	Error in or no operand.
09	Found illegal character in decimal string.
08	Undefined label (may be illegal label).
07	.EN pseudo op missing.
06	Duplicate label.
05	Label missing in .DE or .DI pseudo op.
04	.BA or .MC operand undefined.
03	Illegal pseudo op.
02	Illegal mnemonic or undefined macro.
01	Branch out of range.
00	Not a zero page address.
ED	Error in command input.
2F	Overflow in file sequence count (2**16 max.)
2E	Overflow in number of macros (2**16 max.)
2D	
2C	
2B	.ME without associated .MD
2A	Non-symbolic label in SET pseudo op.
29	Illegal nested definition.
28	
27	Macro definition overlaps file boundary.
26	Duplicate macro definition.
25	Quantity parms mismatch or illegal characters.
24	Too many nested macros (32 max.)
23	Macro definition not complete at .EN
22	Conditional suppress set at .EN
21	Macro in expand state at .EN
20	Attempted expansion before definition.