Copyright Notes

This manual and the object code (contained on floppy disk) is serial numbered and protected by a legitimate copyright. No part of this manual nay be copied or reproduced without the express written permission of the copyright owner, Eastern House Software. You may make a backup copy of the diskette in order to protect your copy of this software. It is though a Federal crime to make a copy of the manual or floppy disk 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 (or backups) of the object media (diskette) 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 supplying 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. Penalties 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 cannot 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.

Finally, an expressed appreciation is given to the purchaser of this software. I 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

Don Earnhardt
2130 Nettlebrook Dr.
Winston-Salem, N. C. 27106

Contents

1. Introduction

This Macro Assembler (ASSM) and Text Editor (TED) resides simultaneously in approximately 11K bytes of memory ($5000-$7C80). The collective assembler and text editor is referred to as MAE. MAE was designed to work with the 48K Apple II or Apple II Plus microcomputer and Disk II.

As mentioned, the MAE object code occupies just over 11K of memory. In addition to this, sufficient memory must be allocated for the text file and label file (symbol table). Approximately 8K 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 ($5000), MAE will set the file boundaries as follows:

- Text File $3000 - $4FFC
- Label File $1800 - $2FFC
- Relocatable Object Buffer $7800
  ($7800 is a buffer provided internally to MAE)

These boundaries leave memory for other language programs at $0800 - $17FC and at $7C80 - $95FC. The label file and text file that MAE generates is position independent and may be located practically anywhere in RAM memory. The object code file location is dependent on the Beginning of Assembly (`.BA`) and the Move Code (`.MC`) pseudo operands (pseudo ops.)

Fear not the accidental reset! The MAE contains reset protection which executes a warm start when the reset key is pressed. This prevents any loss of data if an accidental reset occurs. Additionally, if you exit the MAE to the monitor or to BASIC, the reset key will cause a return to that particular language. In order to return to the MAE, perform a warm start (see Part 11). The previous text file should remain intact.

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. Some unique features of MAE are:

- Coexists with Apple Integer and/or Applesoft BASIC Languages.
- Supports Macro, Conditional, 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.
- Manuscript feature for composing letters and other text.
- Loading and Storing via the Apple Disk II.
- Supports various printer arrangements.
- String search and replace capability, plus other powerful editing commands.
- Capability to send command strings to Apple Disk II.

MAE uses a prompter character: "`)`". This parentheses character is used to indicate that MAE 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 "`)`" proceeding the mnemonic.

Initial entry (or cold start) to MAE is at address $5000. If the break command, "`)BR`", is executed, one may reenter MAE at $5003. Initial entry provides the following default parameters:

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

MAE was designed to coexist with Apple BASIC. This was accomplished by preserving BASIC's zero page variables. Thus, on cold start entry, MAE copies $98 - $FF of zero page to a save area ($7698 - $76PF). On all exits (via "`)BR`", "`)RU`" or "`)US`"), MAE restores these variables. On warm start entry, MAE swaps zero page with the save area. MAE also uses a number of absolute variables at $7700-$77FF.

You should protect MAE from BASIC by setting the BASIC variable HIMEM ($73, $74) to point to Just below MAE and its text and label files. For example, to protect memory above $1800, enter $00 at location $73 and $18 at location $74.

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.

2. Files Contained on the Diskette

The supplied diskette contains the following files:

Filename		Description
`MAE.EXE`		MAE object code
`RELOC.EXE`		Relocating Loader object code
`RELOC.REL`		Relocating Loader relocatable code
`MAE.NOT`	*	Some notes on MAE
`WORDP.EXE`		Word Processor Program
`WORDP.INS`	*	Word Processor Instructions File and example of raw text
`MLMACROS.MLIB`	*	File of some Machine Language Macros
`SWEET16.MLIB`	*	File of SWEET16 Macros
`UART.CTL`	*	Example program control file

UART.M01	*	Example program modules
UART.M02		(UART Driver)
UART.M03		
PARL.OUT.EXE		Parallel printer driver
TEXTOUT.SOURCE	*	Program to convert MAE files to text files
TEXTOUT.EXE		Textfile program

* Indicates source files in MAE format.

3. Text Editor (TED) Features

The TED occupies approximately one-half the total memory space of this 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 Apple II 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 ")" level.
Control+O	Continue processing but suppress output except for errors.
Control+Q	Continue outputting after control S.
Control+C	Exit to Apple monitor. (5003G to re-enter MAE)
Control+B	Exit to BASIC. (Call 20483 to re-enter MAE)

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 a carriage return is given. 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 all 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.

A disk filename may be specified in some of the following commands. Wherever “file” is given as a command parameter, its format is as follows:

Dn “name”

where: n is the disk Drive/Slot number (default - 16 for drive 1, slot 6 - i.e. D16)

“name” is the file name (enclosed in double quotes)

Examples are: D16 “MAE.NOT”
D26 “RELOC.REL”
“DOS SUPPORT”

A description of each command follows:

)ASSEMBLE "file" w

If file is specified, load the file into text area and if no file specified, then assemble contents of text area.

If w = "LIST" then generate a listing. If w = "NOLIST" or 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.

)BASIC

Restore zero page and go to BASIC.

)BREAK

Restore zero page and go to Monitor.

)CLEAR

Clear 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.

)DC "command"

Pass disk commands to Apple DOS. Any DOS command that can normally be entered as a direct command may be passed.

Examples:	Output catalog is:	)DC "CATALOG"
	Delete file TEST is:	)DC "DELETE TEST"

)DC "@filename"

Apple MAE generates two binary files for each source file saved on the disk. These files are represented in the catalog as "filename" and "filename*". The file with the trailing asterisk in its name is used by MAE to calculate the length of the actual source file during loading. MAE uses this as a safeguard to prevent you from loading a file larger than the boundaries allocated for the text file. If the file is larger than allocated memory, MAE will not load it but instead issue a !0F error.

To summarize, the)DC "@filename" command is used when it is desired to delete both entries in the directory (including the filename* file).

)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.

)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 "file" y

Get file from disk and store in the text buffer. If y is not entered, store at start of text buffer. If y is a line number, enter following specified line number. If y = "APPEND" then enter following current contents of text file.

Examples:
```
)GET "MEMTEST"
)GET D16 "CRTDVR" 1000
)GET 027 "UART" APPEND
```

)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 so MAE and the printer are synced.

)LABELS w

Print out the entire contents of the label file if w = "ALL" or w is 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.

)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 "file"

Create a relocatable object file on disk. This command uses the 256 byte relocatable buffer that can be reallocated via the)SET command.

)PASS "file"

Execute second pass of assembly. The first pass must be previously performed. If a filename is entered in the)PASS command, then the text file is loaded before executing the second pass, else)PASS will assume the file is in the text area.

)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 neither x nor y are entered, the entire file is printed.

)PUT "file" 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. (Hint: use this to save your source code)

)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 (ts), end (te), label file start (ls), end (le), and relocatable buffer start (bs)) 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, precede 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).

Caution should be used not to set either the text file or label file boundaries to $800. This conflicts with BASIC.

)TI w n (See Part 9)

Assign terminal input (keyboard) as Apple if w = "APPLE", or serial device if w = "SERIAL". Both input and output will be assigned to the serial device if w = "TERMINAL". If entered, n is the number of pad bits to be sent on occurrence of carriage return.

When **)TI SERIAL** or **)TI TERMINAL** is entered, you must type S on the serial keyboard so MAE can determine the baud rate of the device. Permissible baud rates are 110, 300, 600, 1200, 2400, 4800, 7200, and 9600. After you type S, press the return key. If MAE receives a valid carriage return character, control is then transferred to the serial device. If a valid carriage return is not received, control will remain with the Apple.

)TO w n m (See also Part 9)

Assign terminal output (CRT or printer) as: Apple screen if w = "APPLE", or printer (thru MAE printer driver) if w = "PRINTER". If w = "SERIAL" then output will be directed thru the MAE software UART driver (See Part 9). If w = "ALL" then output will be directed thru both the printer vector and the software UART.

If you are using a standard Apple printer interface, then instead enter ")DC "PR#n"" to turn on your printer (see Part 15).

If you do not have a standard Apple II printer, you may insert a JSR to your printer driver at $791B. Currently this is configured for use with the popular Heath H14 printer.

If w = "SERIAL" or "ALL", then n is the baud rate code and m is the number of pad bits on occurrence of carriage return. The baud rate code (n) is as follows:

n	baud rate
0	110
1	300
2	600
3	1200
4	2400
5	4800
6	7200
7	9600

)USER

Restore zero page and go to location $0000. You must have entered a `JMF` instruction at that address. Note that BASIC defaults $0000 to point to its warm start entry.

b. Entry/Deletion/Change of Text

Source 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 line 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 (via the "`)AU`" command). Any "`)AU x`" command where **x** does not equal 0 puts the TED in the auto line number mode on the next entry of a line number. To exit from this mode, type "`)//`".

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 the "`)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.

4. 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 the "`)OU`" command), 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 user's discretion 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 either of a standard 6502 mnemonic from Table A, a pseudo op from Table B, or a 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 permissible 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 a label with a 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

ADC	BMI	CLD	DEY	LDA	PHP	SBC	NOP
AND	BNE	CLI	EOR	LDX	PLA	SEC	TAX
ASL	BPL	CMP	INC	LDY	PLP	SED	TAY
BCC	BRK	CPX	INK	LSR	ROL	SEI	TSX
BCS	BVC	CPY	INY	CLV	ROR	STA	TXA

BEQ	BVS	DEC	JMP	ORA	RTI	SIX	TXS
BIT	CLC	DEX	JSR	PHA	RTS	STY	TYA

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

TABLE B - Pseudo Ops

.A7

Store ASCII strings in Apple format (bit 7 set). The input format is .A7 'this string is in Apple format', refer to the .BY pseudo op for further details.

.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.

NOTE: When .BA, .MC, and .OS are used in a file, they must be used in this order.

.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
```

In order to insert consecutive blank spaces in an ASCII string using either a .A7 or .BY pseudo op, you must enter two apostrophes and a single space prior to the string. Example:

.A7 "-'----ABC' ("-" represents a space character).

.CE

Continue assembly if errors other than !07, !04, or !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 an .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 the source listing. This is the default state. See Part 4E.

.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 4E.

.FI file

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).

.IN label

Outputs "?" followed with a space and then accepts up to 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 used must be symbolic and should be defined similar to the following example:

```
BEGIN.ADDR
      .PR "ENTER ASSEMBLY START"
      .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.

If the label specified in the .IN pseudo op has not been defined when the .IN is encountered, Mae will not accept input, for example:

```
      .IN VALUE
      …
VALUE
```

In this case, MAE will not accept input. To correct this, simply move the label to or in front of the .IN pseudo op.

.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 4E.

.ME

Macro end of definition. See Part 4E.

.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 an .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 proceeds the numerals then the ASSM assumes decimal (example: 147). If "$" precedes, then hex is assumed (example: $F3). If "%" precedes, 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 recognized:

"+" for addition
"-" for 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
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 la 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 the 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 on disk using `)PU`.

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 "MEM.TEST"`). 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  D16 "X.M01"
.FI  D16 "Y.M02"
.FI  D16 "Z.M03"
.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 passes for a complete assembly. When the end of a pass is encountered, MAE will output the message `END MAE PASS1`. 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 (via `)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 `.SB` stores a two byte external address and the `.SI` stores a two byte internal address. Similarly the relocating loader will alter the internal address and not the external address.

An example of an external address would be the calls to Apple monitor routines (eg. `JSR $FDED`) or any location whose address remains the same no matter where the program is located. Other external addresses would be locations in zero page that the Apple monitor or DOS uses. An example using an internal address would be when you want to store the address of a number of subroutines in a table. An example illustrating how MAE stores command mnemonics and associated processing routine is as follows:

```
.BY 'CO'          ;COPY COMMAND
.SI COPY.MOD      ;COPY MODULE
.BY 'MO'          ;MOVE COMMAND
.SI MOVE.MOD      ;MOVE MODULE
.BY 'PR'          ;PRINT COMMAND
.SI PRINT.MOD     ;PRINT MODULE
```

Expressions consisting of internal and external labels will be combined and considered an internal address. For example, suppose the label `BUFFER` was contained in your program as `BUFFER .DS 80`. Then references consisting of the label `BUFFER` and the external reference `+3`, such as `LDA BUFFER+3`, would be combined and considered .as an internal reference. Thus the relocating loader will alter references to `BUFFER+3` when you relocate the program. 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. (The program will not be relocated properly unless it is assembled using a .BA $2000 statement.) 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. Execution begins at $800 if in the monitor or CALL 2048 if in BASIC. The relocating loader will request the following:

1.	FILENAME?	Name of the file containing the relocatable object code. (Enter with quotes around name.)
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 (via BSAVE) as an executable file, which may be reloaded without using the relocating loader.

As an example, let's assume we want to relocate a program named UART to execute at location $3000, but store the object code at $1000, and start the zero page variables at $0060, and the absolute variables at $4000. We would respond to the relocating loader as follows:

```
FILENAME?           D16 "UART.REL"   File name
Z-PG OFFSET?        60               Assign start of zero page var.
ABS OFFSET?         4000             Assign start of absolute var.
PGM EXE OFFSET?     3000             Program body start
PGM STORE OFFSET?   1000             Store of code start
LOAD MAP
                    …                R. L. outputs a load map
FILENAME?                            Enter just carriage returns 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:

```
BNE SKIP
NOP
…
INCD (VALUE.1)     ;INCREMENT DOUBLE
```

```
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 precede 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:

```
!!!INCD   .MD  (LOC)    ;INCREMENT DOUBLE
          INC  LOC
          BNE  SKIP
          INC  LOC+1
SKIP      .ME
```

This is a possible definition for INCD. The assembler will not produce object code until there is a call for expansion.

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 increments 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 around 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:

```
!!!INCD    .MD   (LOC)      ;INCREMENT DOUBLE
           INC   LOC
           BNE   ...SKIP
           INC   LOC+1
...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 further 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 further 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, let's 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.
`IFF  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 CHAR. PER LINE
           …
           ***
           IFE CHAR.LINE-64 ;CODE FOLLOWS FOR 64 CHAR. PER LINE
           …
           ***
           IFE CHAR.LINE-80 ;CODE FOLLOWS FOR 80 CHAR. 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 of conditional assembly, is the ability to cause a macro to be fully expanded on the first reference and only partly expanded on subsequent references. 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 succeeding 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. Interactive assembly could be used in our CRT controller example as follows:

```
              .PR   "INPUT WHICH CONTROLLER (40, 64, 80)"
CHAR.LINE     .IN   CHAR.LINE
```

NOTE: In the above example, the label CHAR.LINE was defined on the same line as the .IN this is perfectly OK and it eliminates the need to define CHAR.LINE on a separate line using the .DE pseudo op. You might say that it kills two birds with one stone.

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 $0800 (otherwise use .BA)
- Halts assembly on errors (otherwise use .CE)
- Stores object code beginning at $0800 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)

5. Relocating the Relocating Loader

A relocatable object file of the relocating loader is contained on the diskette with the name RELOC.REL.

To relocate the relocating loader, load the executable copy (RELOC.EXE) and begin execution. When FILE NAME? is output, enter "RELOC.REL". Then enter 0 for Z-PG OFFSET? and 0 for ABS OFFSET?. Finally, enter the address of the location you want the relocating loader to execute and reside for PGM EXE OFFSET?, and PGM STORE OFFSET?.

When the relocator completes its task, you may save an executable copy on disk using the Apple monitor. Just remember, execution begins at the address specified for the PGM EXE OFFSET, not $0800 as for the executable copy supplied (RELOC.EXE).

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-existent 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	
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 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 conditional 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 `.HD`
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 parameters 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.

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
```

or

```
)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 the don't care character. Precede with % character to change the don't care character, else the 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 t entered then print all lines altered.

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

```
)EDIT /LOOP/START/ 100 600
```

If you want to delete a reference to LOOP, enter a null string replacement as follows:

```
)ED /LOOP//
```

If you want to delete all references to LOOP01, LOOP02, etc., enter:

```
)ED /LOOP%%//
```

To simply delete all occurrences of LOOP, enter:

```
)EDIT /LOOP// 100 600
```

To do the same thing as above but not display changes on the screen, enter:

```
)ED /LOOP// # 100 600
```

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

Suppose, for example, there are some occurrences of the label BUFFER that you wish to change to CRT/BUFFER, some you wish to delete (the entire line), and some that you wish to extensively edit. This requires a conditional replacement capability with you as a decision maker. To do this, enter:

```
)ED &BUFFER&CRT/BUFFERS *
```

(Note that the string terminator & is used because / is actually a part of the string.)

Now, each time MAE's edit command encounters the string BUFFER, the line will be printed on the screen and an asterisk (*) prompter will appear awaiting your entry of one of the subcommands (A, D, M, S, X or 2) described above. For example, to replace BUFFER with CRT/BUFFER, press "A". To delete the entire line, press "D", to extensively edit the line, press "2", etc. When you have made your decision for the current line, the next occurrence of BUFFER will be displayed. This continues until the end of the text file or until you enter the "X" subcommand.

At the end of the)EDIT operation, the number of occurrences 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 this command, the cursor will be positioned at the beginning of the line. The usual Apple screen editing escape characters may be used to change the line of text. When the return key is pressed, MAE will insert the corrected text into the text file.

b. Find Command

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

```
)FIND tS1t # x y
```

Where: t, S1, #, x, and y are as defined in the)EDIT command.

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

At the end of the)FIND operation, the number of occurrences 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. Control Codes (Serial Device)

ASCII characters whose hex values are between hex 00 and 20 are normally non-printing characters. With a few exceptions, these characters will be output in the following manner: C, where C is the associated printable character if hex 40 was added to its value. For example, ASCII 03 will be output as C, and 18 as X, etc.

In addition, some of these control codes have special functions in MAE.

Control codes which have special functions are:

Code		Description
@		Null (hex 00)
B		Restore zero page and go to BASIC
C		Restore zero page and go to Monitor
G	*	Bell
H	*	Backspace (delete previously entered char.)
I	*	Horizontal tab to next 8-th char, position
J	*	Line feed
M	*	Carriage return
O		Continue processing but no output
Q	*	Continue after stop via break key
X		Delete entire line altered
Y		Restore zero page and jump to location $0000. (you may reenter at $5003)
Z		Terminate processing and go to ")" level
[	*	Escape character

* = Non-printing control character.

9. Using MAE's Built-in Software UART

NOTE: This is an optional feature of MAE and need not be incorporated.

If you want to connect an 80 column device (such as an external CRT terminal or TTY) to your Apple because the 40 column screen seems "cluttered", you can use MAE's built-in software UART driver. This software UART is also useful if you want to "remote" the Apple and access via a terminal and modem. All that is required is a parallel port card, you don't have to purchase a special serial I/O card for this application.

Practically any serial device may be connected and controlled by this software. This software UART converts bytes to serial format and inputs and/or outputs using just 2 bits of a parallel port. The data format consists of one start, 8 data, and 2 stop bits. Via appropriate commands, the user can specify the baud rate and also the number of pad bits on carriage return.

We have found that many RS232 devices can be connected directly to a TTL level parallel port without level conditioning circuitry. But if you provide RS232 or other conditioning circuitry, you should not invert the signals as they are in their positive true state.

The commands)TI and)TO are provided to direct MAE to input or output thru this software UART, and to specify baud rate and pad bits.

Examples:	)TO SERIAL n a	direct output thru UART transmitter
	)TI SERIAL n m	input from UART receiver
	)TI TERMINAL n m	direct all input and output thru software UART

Note:)TI SERIAL and)TI TERMINAL will "lock out" the Apple keyboard. To revert back to the Apple keyboard, type)TI APPLE on the external terminal keyboard.

The software UART is shipped un-configured. To activate this software you must configure it for your system by supplying appropriate addresses and bit positions as follows:

```
Addr. Obj. Code   Source

                  ;IF PIA THEN INITIALIZE DIRECTION REGISTER
                  ;ELSE JUST NOP THIS PART
7276 AD xx xx        LDA PORT.DIR   ;GET FROM DIRECTION REGISTER
7279 29 xx           AND #xx        ;INITIALIZE BIT FOR INPUT
727B 09 xx           ORA #yy        ;INITIALIZE BIT FOR OUTPUT
727D 80 xx xx        STA PORT.DIR   ;STORE IN DIRECTION REGISTER
                  ;GET SERIAL INPUT FROM PARALLEL PORT
                  ; (SAME CODE BUT THREE DIFFERENT LOCATIONS)
7285 AD xx xx        LDA PORT.DATA  ;GET FROM PORT
7288 29 xx           AND #zz        ;MASK OUT ALL EXCEPT INPUT BIT
728E AD xx xx        LDA PORT.DATA  ;GET FROM PORT
7291 29 xx           AND #zz        ;MASK OUT ALL EXCEPT INPUT BIT
72B9 AD xx xx        LDA PORT.DATA  ;GET FROM PORT
72BC 29 xx           AND #zz        ;MASK OUT ALL EXCEPT INPUT BIT
                  ;PUT SERIAL OUTPUT TO PARALLEL PORT
7344 AD xx xx        LDA PORT.DATA  ;GET FROM PORT
7347 29 xx           AND #ww        ;DEFAULT TO OUTPUT ZERO BIT
7349 90 02           BCC SKP        ;BRANCH IF TO OUTPUT ZERO BIT
734B 09 xx           ORA #vv        ;ELSE OUTPUT A ONE BIT
734D 8D xx xx SKP    STA PORT.DATA  ;PUT TO PORT
```

10. Examples

a. TED Examples

#1 Illustrate ways to load MAE and begin execution at the cold start:

```
BLOAD MAE.EXE or BRUN MAE.EXE

CALL 20480
```

#2 Illustrate entry of text:

```
)AUTO 10
```

```
)1000 ;THIS IS A TEST
1010  LOOP    LDA  VALUE,Y
1020          NOP
1030  END.PGM .EN
1040//
```

Notes: enter "//" to exit; uses auto line numbering

#3 Illustrate listing of text:

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

#4 Put file to disk (drive 2, slot 6) with name TEST:

```
)PUT D26 "TEST"
```

#5 Get file from disk (drive 1 slot 6) named TEST:

```
)GET "TEST"
```

Note: The default is drive 1 slot 6.

#6 Assemble file CRTDVR and generate a listing:

```
)ASSM "CRTDVR" LIST
```

#7 Output directory for default drive then for drive 2, slot 6:

```
)DC "CATALOG"
)DC "CATALOG, D2.S6"
```

#8 Delete binary file TEST.EXE:

```
)DC "DELETE TEST.EXE"
```

#9 Delete source file TEST.ASM:

```
)DC "@TEST.ASM"
```

Note: actually deletes TEST.ASM and TEST. ASM*

#10 Clear the text file:

```
)CLEAR
```

#11 Direct output to printer:

`)TO PRINTER` - with non-standard interface or standard interface using PARL.OUT driver (included))

`DC "PR#n"` - with standard interface at slot n not using PARL.OUT

#12 Direct output thru UART at 300 baud with 10 pad bits:

```
)TO SERIAL 1 10
```

#13 Assign device associated with UART as input (keyboard) and output (with 2 pad bits):

```
)TI TERMINAL 2
S
```
-user types S on serial keyboard then types RETURN.

#14 Find all occurrences of the text LDA:

```
)FIND /LDA/
```

#15 Replace all occurrences of "LDA FA" with "LDA *FA" between lines 1000 and 2000:

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

#16 Provide for 15 characters per label:

```
)FORMAT SET 15
```

#17 Output all fixed (external) labels:

```
)LABELS FIXED
```

#18 Renumber the text file beginning at line number 100 and incrementing by 5:

```
)NUMBER 100 5
```

#19 Move lines 100 thru 200 to after line 9000:

```
)MOVE 9000 100 200
```

#20 Print lines 900 thru 976:

```
)PRINT 900 976
```

#21 Reallocate the text file to $400 thru $1FFC:

```
)SET $400 $1FFC
```

#22 Go to BASIC:

)BASIC or CTRL+B

Note: return via CALL 20480 (cold start) or CALL 20483 (warm start)

#23 Go to Machine Language Monitor:

)BREAK or CTRL+C

Note: return via 5000G (cold start) or 5003G (warm start)

#24 Run assembly program at symbolic label BOX:

```
)RUN BOX
```

#25 Run assembly program at hex address $A03:

```
)RUN $A03
```

b. ASSM Examples

#l Begin assembly at $1000 and store object code:

```
.BA $1000
.OS
```

#2 Begin assembly at $1000 but store object code at $4000:

```
.BA $1000
.MC $4000
.OS
```

#3 Define the CRT output routine:

```
COUT .DE $FDED
```

#4 Assign an internal work location in zero page:

```
WORK .DI $0
```

#5 Allocate 6 bytes of storage:

```
TABLE .DS 6
```

#6 Define label E0I as a mask with bit 6 set and show its use in an AND statement:

```
E0I .DE %01000000
 AND #E0I
```

#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 an example of the .BY pseudo op:

```
.BY 'ALARM CONDITION ON MOTOR 1' $0D $0A
```

#9 Store the address of the internal label TABLE and the external label COUT:

```
.SI TABLE
.SE COUT
```

#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 the)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 the)FO command) are allowed if defined on a line with no mnemonics.

#12 Reference the call to the BASCALC Apple routine so the relocating loader will not alter the address during loading:

```
BASCALC   .DE  $FBC1
          ...
          JSR BASCALC
     — or —
          JSR $FBC1
```

11. Getting Started with MAE

Load MAE as follows:

1) Insert MAE diskette in appropriate disk drive.

2) Boot the system from the MAE diskette.

Do if you have Applesoft ROM BASIC:

3) A menu will be displayed on the screen.

   Select from the menu to execute MAE or the Relocating Loader, or to go to BASIC or the Monitor. To get started, select to execute MAE.

Do if you have Integer ROM BASIC:

4) The message FILE TYPE MISMATCH will be displayed because the greeting program was in Applesoft BASIC.

   (Note: A listing of this menu program is provided in Part 16 which you can modify for Integer BASIC if necessary.)

   Simply BRUN the desired program. To get started, type BRUN MAE.EXE

MAE will respond with:

```
C 1979 BY C MOSER
3000-4FFC 1800-2FFC 7800
3000      1800
```

This displays the default allocations of memory for the text file ($3000-$4FFC), label file ($1800-$2FFC), and start address of the 256 byte relocatable buffer ($7800). 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.

Remember, to exit MAE, issue either the)BASIC or)BREAK commands to go to BASIC or to Monitor. You may reenter MAE via 5003G (warm start - everything preserved) or 5000G (cold start - everything cleared to default state). If you are in BASIC, type CALL 20483 to warm start MAE. This is the same as 5003G from the monitor.

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

```
)GET "MAE.NOT"
)FORMAT CLEAR    turn formatting off
```

The MAE.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 3, assembler features in Part 4, and then the examples in Part 10.

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

12. The 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 STP coexists with the MAE and occupies memory between $A00 and $152F. The primary purpose of this word processor was to provide a simplified means to process program documentation and for other text processing needs. (This manual was produced using the STP.) This simplicity was accomplished with a set of easily remembered word processing functions, and usage of an already familiar text editor to enter and edit the raw text.

STP, unlike some word processing programs, can output the formatted text to the screen. This, is most useful on 80 column displays and can result in a tremendous savings in time and paper.

The STP word processor also provides lower case printing capability (for printer only) for Apple II systems without lower case hardware modifications. This, of course, assumes that the printer is capable of printing lower case characters. The STP automatically converts all alphabetic characters to lower case for the printer and then back to upper case for the CRT. Two special command characters are used to provide upper case: "]" and "<". These characters are inserted in the raw text file but are not printed by either the printer or the CRT when the file is formatted. In order to provide a single upper case character at the beginning of a sentence or the first letter of a proper name, use the "<" Immediately before typing the character to be capitalized. The "]" character is used like a shift-lock key. When the STP first recognizes this special character, the lower case conversion is deactivated until another "]" character is encountered. The printer

driver Included in the MAE (the nonstandard serial driver or the PARL.OUT driver included on the MAE disk) is required in order to utilize the "<" and "]" characters to obtain lower case characters on the printer.

If your system has lower case modifications installed, the STP can be configured to handle lower case to the Apple screen by inserting three NOPs at $0B3D-$0B3F and at $1485-$1487.

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 left.

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. This means that the lines will be printed without adding spaces to make the margins come out even. Also, lines of text are not combined to fill to the specified margins.

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.

Conditional Form Feed (`.ff n`)

Perform a form feed if less than `n` lines remain at the bottom of the page.

Literal Space (^ character)

Normally, spaces are not processed like other characters. If several spaces are entered consecutively, the STP word processor recognizes 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 but one must be in the shift-lock mode. The caret will not be printed when the text is processed but instead a space will be printed for each occurrence of the caret.

Turn on Justification (`.ju`)

The `.ju` macro may be entered in order to restore justification. `.ju` is normally used to revert back to Justification after 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. Lines of text 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 `.l2`. If you enter `.l` by itself, one will be assumed. Thus `.l` and `.l1` 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.

Paragraph Numbering or Identification (`.pi text`)

The best use for this macro is for paragraph numbering. An example for this is "`.pi 1.01`".

Page Title (`.tl 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 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.

File Linkage (`.link filename`)

This macro can link text modules together as a single printable document. An example is `.link d26 "SECTION14"`. Simply enter the link macro at the bottom of the file immediately before the one you want loaded and processed. Thus the first file would have a `.link` to the second, the second to the third, etc. There is no limit to the number of files which can be linked.

Running Machine Language Programs (`.ru $xxxx`)

This macro allows the user to run a machine language program at any point during the text formatting process. This is helpful when it is desired to send command strings to the printer to change the character set, to perform elongated printing, etc. The machine language program issues the command string and performs an RTS back to the MAE. The text processing then continues with no indication of the interruption.

Printing Text (`)WC` and `)WP`)

Normally `)RU $A00` Is used to send STP output to the CRT, and `)RU A03` is used to send the output to the printer. The STP also contains an initialization routine at `$A06` which provides two new commands. After the initialization routine is run by typing `)RU`

$A06, the)WP command can be used to send STP output to the printer and)WC can be used to send It to the CRT.

When the)WP and)WC commands are performed, the .link macros are ignored unless specifically asked for by typing)WP LI. Additionally, the first a pages may be suppressed by typing)WP LI n or)WP n. These options can also be used with the)WC command.

Creating Shape Tables (.shape n and .set n l 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 l p are as follows:

- n line number for this margin specification
- l 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 the Apple's 40 column screen.

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

a. How to Use the STP Word Processor

1) Load the word processor and MAE via:

```
BRUN MAE.EXE
)DC "BLOAD WORDP.EXE"
```

2) Clear format mode via `)FORMAT CLEAR`.
3) Enter raw text using MAE for editing. Include all necessary text processing macros.
4) When you are finished entering the raw text and associated text macros, generate a formatted output via:

`)RUN $A00`	for output to CRT only
`)RUN SA03`	for output to CRT and Printer
`)RUN $A06`	to Initialize output commands `)WP` and `)WC`. Subsequently, `)WP` sends STP output to the printer, and `)WC` send It to the CRT. The Parallel or Serial Printer Driver in MAE, is required for this mode of operation.

b. Example

A raw text file named `WORDP.INS` is contained on the diskette. Type `)GET "WORDP.INS"` to load this file. Type `)PRINT` to examine the raw text with associated macros. Type `)RUN $A00` (CRT only) or `)RUN $A03` (CRT and non-standard Printer) to execute the word processor and output the text in word processor format. Note: If you output this to a 40 column Apple, It will not appear neat since the margin was set for 76 characters per line.

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.

13. Special Notes

a. Converting Eastern House ASSM/TED Files to MAE Files

1) Use usual procedure to enter MAE.
2) Use `)SET` to allocate the necessary space for file to be converted.
3) Use `)BR` to enter monitor.
4) Clear text file area as follows.

`*3000:0`	(first byte of text file - 00)
`*3001<3000.4FFCM`	(all bytes - 00)

Note that this example uses boundaries $3000 - $4FFC. This step should be changed to agree with your boundary settings.

5) Warm start MAE (*5003G)
6) BLOAD old ASSM/TED file from disk

)DC "BLOAD Snn.file,A$xxxx" (xxxx = current start of text file)

7) Type)PR / to list last line.
8) Return to monitor with)BR.
9) Move contents of $76E4 and $76E5 to $76DA and $76DB.
10) Warm start MAE (*5003G).
11) Save new MAE file using)PUT command.

b. Editing Basic Programs with MAE

Wouldn't it be nice to be able to use the copy, move, search/replace, and auto line numbering features of the MAE to edit BASIC programs? Well, it can be done.

The editor cannot tell the difference between assembly language, text such as you are now reading, and BASIC instructions. However, the Apple DOS EXEC command is the thing that makes this idea possible. The text can be created with MAE's editor and then saved as a text file on the disk. When the program is to be executed, simply type EXEC "textfilename" from BASIC and the program will be loaded from disk and executed. Before running the program in this manner, you should type NEW in order to remove any BASIC program currently residing in RAM. If this is not done, the text file program will be appended to or overlaid on the existing program, depending on the line numbers of the two programs.

The TEXTOUT.EXE program is used to create the text file when the BASIC program has been completed and is still in the MAE text area (the text file boundaries must begin at $3000). You just type)DC "BLOAD TEXTOUT.EXE" and then)RU $805. The TEXTOUT.EXE program will ask you for a name for the output text file (it must be limited to 20 characters). When the name has been entered and followed by a RETURN, the TEXTOUT.EXE program will begin loading the text from MAE's text area into a sequential text file on the disk. The line numbers are converted from the packed- BCD form (low byte first) used by the MAE, to ASCII and stored with the most significant digit stored first. Additionally, leading zeros of the line number are deleted. The Instruction is then stored until the end-of-line byte is found (this is determined by testing bit 7 for 1).

After each line is stored, the next line is tested for end-of-file. If the third byte of the line (first byte following the line number) is 00, then all data has been saved and the text file can be closed. The TEXTOUT.EXE program will then return to the MAE. You may then type)BA or CTRL+B to exit to BASIC in order to test the text file program.

The text file can be reloaded into the MAE by typing)DC "EXEC filename". This allows for subsequent editing of the source program.

When the BASIC program is complete and tested, it can be saved as a regular BASIC program or RUN once it has been EXECed into RAM from BASIC.

c. Converting MAE Files To and From Other Assemblers

The TEXTOUT program can also be used to transfer MAE assembler files to another assembler which requires text-file source code. In order to transfer source files from another assembler to the MAE, the text file created by the other assembler can be EXECed into the MAE as previously described. If the other assembler does not store line numbers in the source file, the other assembler must be used to initiate auto line numbering in the MAE. This is done by making the first line of the file "auto 10" and the second line "0" (the quotes are for emphasizing the line, they should not actually be typed in the text file).

d. Miscellaneous 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.
- MAE's internal software UART resides in locations $7303-$7406.
- An 80 character/line output device should be used when printing an assembly listing In order to provide a neat printout without fold-over to the next line.
- If you are going to use MAE and BASIC together, alter HIMEM ($0073, $0074) so BASIC will not clobber MAE, or its text or label files.
- Use quality diskettes like Scotch or Dysan. A few dollars saved on a cheap diskette is not worth the risk of lost data.
- 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.MLIB	Library of Macros
name.DOC	Program Documentation
name.INS	User Instructions
name.NOT	Program Notes
name.BAS	BASIC Program
name.DAT	BASIC Data File

- The MAE can be configured for use with 80-column cards such as the Sup'R'term card. If you have one of these cards, make the following patches to the MAE and then resave It (BSAVE MAE,A$5000,L$2C90)
  (1) Insert NOPs ($EA) in MAE locations $7A4B-$7A4D.
  (2) At $7A55 insert :20 60 7C (20 80 7C? or 20 90 7C?)

(3) At $7C90 insert :A9 8C 20 ED FD A9 98 20 ED FD 60 This provides the HOME command for Sup'R'tem but may also be used for other 80-column cards.
(4) At $68BE insert :4E

Note: You may also want to make the changes to the STP as shown in Part 12 so that the 80-column/lower case card can be used for text processing.

14. ASSM/TED Users Goup

An ASSM/TED Users Group has been formed by James Strasma for the exchange of programs and unique modules. Most of the information in this exchange is oriented to PET MAE but should soon contain some Apple MAE compatible software. The cost per diskette is also minimal but the Information Is extremely useful. For more details, contact:

James Strasma
c/o Grace U.M.C.
120 West King Street
Decatur, 111. 62521

15. How to Configure for Printer

If you are using an Apple parallel or Centronics or Tymac parallel printer card, you will want to change the printer driver internal to the MAE. This can be done very easily by typing)AS "PARL.OUT" . This file is included on the MAE disk and can be modified to use the interface card in any slot (see comments in PARL.OUT file). If you are using the card in slot number 1, no modification is required. In that case, all you need to do is type)DC "BLOAD PARL.OUT.EXE". In either case, you should resave the MAE when the printer driver has been loaded or assembled. This is done by typing)DC "BSAVE MAE,A$5000,L$2C90". When this change has been made, the printer can be activated by typing)TO P and deactivated by typing)TO A. Also,)RU $A03, or)WP can then be used to send STP output to the printer.

If you are using a non-standard printer Interface (i.e. one which does not work via PR#n), you may either use the provided non-standard printer driver or substitute your own. In any case to turn on the non-standard printer, type)TO PRINTER. To turn off, type)TO APPLE.

The non-standard printer driver provided within MAE is designed to be utilized with an RS232 parallel to serial interface card such as the one provided by Electronic Systems Co., and the Heathkit H14 printer. This printer driver is set up to operate with the serial interface card in slot 0. Just in case this is not compatible with your system, we have provided a listing of the printer driver which you can use reconfigure for your requirements. The listing is provided in Part 16.

16. Example Listings

A listing of the Applesoft menu program, non-standard printer driver, and an example UART program in MAE's syntax is contained in this section.

The UART program is a software UART driver contained on the supplied diskette under files: `UART.CTL`, `UART.M01`, `UART.M02`, and `UART.M03`.

The UART driver program has three entry points:

1. `SET.BAUD` Optional entry used to automatically measure user terminal baud rate.
2. `UART.OUT` Output character in `R(A)`.
3. `UART.IN` Input character and return in `R(A)`.

Note: The UART program is free to use by MAE purchasers for any non-commercial purpose. For commercial use, we only request that you briefly write describing the use of the UART program. We request no monetary payment or any other remuneration.

a. UART Program

```
                0010            .CT    ;DESIGNATE AS CONTROL FILE
                0020
                0030            .CE    ;CONTINUE IF ERRORS
                0040
                0050            .BA $2000
                0060
                0070 ;        +++++++ DEFINITIONS +++++++
                0080
                0090 PIA.PORT   .DE $E841  ;PIA DATA PORT
                0100 PIA.DIR    .DE $E843  ;PIA DIRECTION PORT
                0110
                0120 MSK.IN     .DE %01000000    ;INPUT IS ON BIT 6
                0130 MSK.OUT    .DE %10000000    ;OUTPUT IS ON BIT 7
                0140
                0150
                0160 ;UART CONTROL PARAMETERS:
                0170 ;----------------------------------------
                0180
2000-           0190 NO.PADBITS .DS 1  ;NO. OF PAD BITS ON CR LF
                0200
2001-           0210 BIT.TIME   .DS 1  ;BAUD RATE CODE (0-7)
                0220 ;                     ;---- ---- ---- -----
                0230 ;                     ;0 =  110  : 4 = 2400
                0240 ;                     ;1 =  300  : 5 = 4800
                0250 ;                     ;2 =  600  : 6 = 7200
                0260 ;                     ;3 = 1200  : 7 = 9600
                0270
                0280
                0290
                0300            .FI D16 "UART.M01"     ;SET BAUD AND TABLE
DELAYS


07F6  432D-4B23     UART.M01

                0010
                0020 ;     +++++ SET BAUD RATE +++++
                0030
```

```
2002- 08       0040 SET.BAUD   PHP               ;SAVE PSR
2003- 78       0050            SEI               ;CLEAR INTERRUPTS
2004- AD 43 E8 0060            LDA    PIA.DIR    ;INITIALIZE PORT ON LOGON
2007- 29 BF    0070            AND    #$FF-MSK.IN     ;       *
2009- 09 80    0080            ORA    #MSK.OUT        ;       *
2008- 8D 43 E8 0090            STA    PIA.DIR         ;       *
               0100
200E- 20 47 20 0110  LP1        JSR GET.BIT
2011- D8 FB    0120             BNE LP1         ;BR. IF ALREADY SPACING
2013- AD 41 E8 0130  LP2        LDA PIA.PORT
2016- 29 40    0140             AND #MSK.IN
2018- F0 F9    0150             BEQ LP2         ;BR. IF MARKJNG
201A- A0 00    0160             LDY #00         ;CLEAR FOR DELAY FACTOR
201C- AD 41 E8 0170  LP3        LDA PIA.PORT    ;GET BIT
201F- 29 40    0180             AND #MSK.IN     ;       #
2021- F0 07    0190             BEQ GOT.COUNT
2023- C0 FF    0200             CPY #$FF
2025- F0 F5    0210             BEQ LP3
2027- C8       0220             INY
2028- D8 F2    0230  SKP.FF     BNE LP3
               0240
202A- 98       0250  GOT.COUNT  TYA             ;MOVE COUNT TO R(A)
202B- A0 00    0260             LDY #00
2820- 09 3F 20 0270  LP.F1      CMP TBLBAUD,Y
2030- B0 03    0280             BCS GOTBAUD
2032- C8       0290             INY
2033- D0 F8    0300             BNE LP.F1
2035- 8C 01 20 0310  GOTBAUD    STY BIT.TIME    ;STORE BAUD RATE CODE
2038- A2 0C    0320             LDX #12         ;WAIT UNTIL ALL BITS HAVE
203A- 20 BE 20 0330             JSR PAD.DELX
203D- 28       0340             PLP
203E- 60       0350             RTS
               0360
               0370
203F- FF       0380  TBLBAUD    .BY 255         ; )= 110
2040- 93       0390             .BY 147         ; )= 300
2041- 4A       0400             .BY 74          ; )= 600
2042- 25       0410             .BY 37          ; )= 1200
2043- 12       0420             .BY 18          ; )= 2400
2044- 0A       0430             .BY 10          ; )= 4800
2045- 07       0440             .BY 7           ; )= 7200
2046- 00       0450             .BY 0           ; )= 9600
               0460
               0470
2047- AD 41 E8 0480  GET.BIT    LDA PIA.PORT    ;GET KEYBOARD INPUT
204A- 29 40    0490             AND #MSK.IN     ;       *
204C- 60       0500             RTS
               0510
204D- AD 01 20 0520  DEL0.5     LDA BIT.TIME
2050- 18       0530             CLC
2051- D8       0540             CLD
2052- 69 08    0550             ADC #08
2054- A8       0560             TAY
2055- 4C 5B 20 0570             JMP EN0.5
               0580
2058- AC 01 20 0590  DLYFULL    LDY BIT.TIME
205B- B9 66 20 0600  EN0.5      LDA UD.TBL1.Y
205E- F0 16    0610             BEQ NOT.THIS
2060- A8       0620             TAY
2061- 88       0630  LOOPDEL1   DEY
```

```
2062- D0 FD    0640            BNE LOOPDEL1
2064- EA       0650            NOP
2065- 60       0660            RTS
               0670
               0680  ;---DELAY=5X+19
2066- 00       0690  UD.TBL1    .BY 00          ;DELAY FULL FOR 110 BAUD
2067- 00       0700            .BY 00          ;DELAY FULL FOR 300      *
2868- 00       0710            .BY 88          ;DELAY FULL FOR 600      *
2069- 9A       0720            .BY 154         ;DELAY FULL FOR 1200     *
206A- 47       0730            .BY 71          ;DELAY FULL FOR 2400     *
206B- 1D       0740            .BY 29          ;DELAY FULL FOR 4800     *
206C- 0F       0750            .BY 15          ;DELAY FULL FOR 7200     *
206D- 08       0760            .BY 88          ;DELAY FULL FOR 9600     *
               0770
               0780  ;---DELAY=5X+28
206E- 00       0790            .BY 88          ;DELAY 0.5 FOR 118 BAUD
206F- 00       0880            .BY 88          ;DELAY 0.5 FOR 300   *
2078- 00       0810            .BY 88          ;DELAY 0.5 FOR 600   *
2071- 48       0820            .BY 72          ;DELAY 0.5 FOR 1200  *
2072- 1F       0830            .BY 31          ;DELAY 0.5 FOR 2400  *
2073- 0A       0840            .BY 18          ;DELAY 0.5 FOR 4800  *
2074- 03       0850            .BY 83          ;DELAY 0.5 FOR 7200  *
2075- 01       0860            .BY 81          ;DELAY 0.5 FOR 9600  *
               0870
2076- B9 8C 20 0880  NOT.THIS   LDA UD.TBL2,Y
2079- A8       0890            TAY
207A- 48 48 48 0900  LOOPDEL2   .BY $48 $48 $48 $48 $48 $48 $48
207D- 48 48 48
2080- 48
2081- 68 68 68 0910  .BY $68 $68 $68 $68 $68 $68 $68
2084- 68 68 68
2087- 68
2088- 88       0920            DEY
2089- D0 EF    0930            BNE LOOPDEL2
208B- 60       0940            RTS
               0950
               0960   ;---DELAY=54X+22
208C- A7       0970   UD.TBL2   .BY 167         ;DELAY FULL FOR 110 BAUD
208D- 3C       0980            .BY 60          ;DELAY FULL FOR 300  *
208E- 1E       0990            .BY 30          ;DELAY FULL FOR 600  *
208F- 00 00 00 1000            .BY 00 00 00 00 00
2092- 00 00
               1010
               1020   ;---DELAY=54X+31
2094- 53       1030            .BY 83          ;DELAY 0.5 FOR 110 BAUD
2095- 1E       1040            .BY 30          ;DELAY 0.5 FOR 300  *
2096- 0E       1050            .BY 14          ;DELAY 0.5 FOR 600  *
               0310   .FI D16 "UART.M02"       ;UART OUTPUT DRIVER

0343  432D-4670       UART.M02

               0010
               0020   ;     +++++ UART OUTPUT +++++
               0030
2097- 08       0040   UART.OUT  PHP
2098- 78       0050            SEI
2099- 20 9E 20 0060            JSR UART.OUT1
209C- 28       0070            PLP             ;RESTORE PSR AND RETURN
209D- 60       0080            RTS
               0090
```

```
209E- 48       0100 UART.OUT1   PHA             ;SAVE CHAR.
209F- 49 FF    0110             EOR #$FF        ;INVERT
20A1- 48       0120             PHA
20A2- A2 0B    0130             LDX #11         ;11 BITS: 1 STOP, 8 DATA,
20A4- 38       0140             SEC
20A5- 20 D2 20 0150 LP.UOUT     JSR BIT.OUT     ;BIT TO PORT
20A8- 20 58 20 0160             JSR DLYFULL     ;DELAY FULL BIT TIME
20AB- 68       0170             PLA             ;RESTORE R(A)
20AC- 4A       0180             LSR A           ;NEXT BIT
20AD- 48       0190             PHA             ;AND SAVE
20AE- CA       0200             DEX
20AF- D0 F4    0210             BNE LP.UOUT     ;LOOP
20B1- 68       0220             PLA             ;REMOVE JUNK
20B2- 68       0230             PLA             ;RESTORE CHAR.
20B3- 29 7F    0240             AND #$7F        ;CLEAR BIT 7
20B5- C9 0D    0250             CMP #$0D        ;CR
20B7- F0 8B    0260             BEQ PAD.DEL
20B9- C9 0A    0270             CMP #$0A        ;LF
20BB- F0 07    0280 BEQ         PAD.DEL
20BD- 68       0290             RTS
               0300
20BE- 48       0310 PAD.DELX    PHA
20BF- E0 00    0320             CPX #00
20C1- 4C C8 20 0330             JMP PAD.DELEN
               0340 ;
20C4- 48       0350 PAD.DEL     PHA             ;PRESERVE
20C5- AE 00 20 0360             LDX NO.PADBITS  ;GET # OF PAD BITS
20C8- F0 06    0370 PAD.DELEN   BEQ EX.DEL      ;SKIP IF ZERO
20CA- 20 58 20 0380 LP.PDEL     JSR DLYFULL     ;DELAY
20CD- CA       0390             DEX
20CE- D0 FA    0400             BNE LP.PDEL     ;LOOP
20D0- 68       0410 EX.DEL      PLA             ;RESTORE
20D1- 60       0420             RTS
               0430
20D2- AD 41 E8 0440 BIT.OUT     LDA PIA.PORT    ;PUT BIT
20D5- 29 7F    0450             AND #$FF-MSK.OUT
20D7- 90 02    0460             BCC SKP.BOUT
20D9- 09 80    0470             ORA #MSK.OUT
20DB- 80 41 E8 0480 SKP.BOUT    STA PIA.PORT
20DE- 60       0490             RTS
               0500
               0510
               0320 .FI D16 "UART.M03"         ;UART INPUT DRIVER


0248  432D-4575     UART.M03

               0010
               0020 ;     +++++ UART INPUT +++++
               0030
20DF- 08       0040 UART.IN     PHP
20E0- 78       0050             SEI
20E1- A9 00    0060             LDA             ;CLEAR CHR.
20E3- 48       0070             PHA ;       *
20E4- 20 47 20 0080 LP.UI1      JSR GET.BIT     ;GET BIT
20E7- D0 FB    0090 BNE         LP.UI1 ;LOOP UNTIL NO BIT
               0100
20E9- 20 47 28 0110 LP.UI2      JSR GET.BIT     ;GET BIT
20EC- F0 FB    0120             BEQ LP.UI2      ;LOOP UNTIL START BIT
               0130
```

```
20EE- 20 4D 20 0140             JSR DEL0.5      ;DELAY UNTIL MIDDLE OF STA
20F1- 20 47 20 0150 LP.UI3      JSR GET.BIT     ;GET BIT
20F4- 38       0160             SEC             ;ASSUME SPACE
20F5- D0 01    0170             BNE SKP.UI1
20F7- 18       0180             CLC             ;NO IT IS MARK
20F8- 68       0190 SKP.UI1     PLA
20F9- 6A       0200             ROR A           ;ROTATE RIGHT INTO CARRY
20FA- B0 07    0210             BCS DONE.UI
20FC- 48       0220             PHA
20FD- 20 58 20 0230             JSR DLYFULL     ;DELAY UNTIL MIDDLE OF HEX
2188- 18       0240             CLC
2181- 90 EE    0250             BCC LP.U13      ;LOOP FOR NEXT BIT
2183- 49 FF    0260 DONE.UI     EOR #$FF        ;INVERT
2185- 29 7F    0270             AND #$7F        ;CLEAR BIT 7
2187- 28       0280             PLP             ;RESTORE PSR AND RETURN
2188- 60       0290             RTS
               0330
               0340
               0350
               0360 END.PGM     .EN
END OF MAE PASS!

---  LABEL FILE:  ---

 BIT.OUT =2002       BIT.TIME =2001    DEL8.5 =204D
 DLYFULL =2058       DONE.UI =2103     EN8.5 =205B
 END.PGM =2189       EX.DEL =20D0      GET.BIT =2047
 GOT.COUNT =202A     GOTBAUD =2035     LOOPDEL1 =2061
 LOOPDEL2 =207A      LP.F1 =202D       LP.PDEL =20CA
 LP.UI1 =20E4        LP.UI2 =20E9      LP.UI3 =20F1
 LP.UOUT =20A5       LP1 =200E         LP2 =2013
 LP3 =201C           MSK.IN =0040      MSK.OUT =0080
 NO.PADBITS =2081    NOT.THIS =2076    PAD.DEL =20C4
 PAD.DELEN =20C8     PAD.DELX =20BE    PIA.DIR =E843
 P1A.PORT =EB41      SET.BAUD =2002    SKP.BOUT =20D8
 SKP.FF =2028        SKP.UI1 =20F8     TBLBAUD =203F
 UART.IN =20DF       UART.OUT =2097    IWRT.OUT1 =209E
 UD.TBL1 =2066       UD.TBL2 =208C

//88B8,2189,21if
)
```

b. MAE Supplied Nonstandard Printer Driver

(Designed for Electronic Systems Serial Interface and Heath H14 Printer)

```
               3920 ;
               3930 ;
7BAD- 48       3940 PR.OUT     PHA             ;SAVE THE CHAR.
76AE- AD 80 C0 3950 WAIT1      LDA $C080       ;CHECK RS-232 INPT FOR BUSY
7BB1- 29 03    3960            AND #$03        ;MASK OUT HI NIBBLE
7BB3- C9 01    3970            CMP #$01        ;($11=NOT BUSY)
7BB5- D0 F7    3980            BNE WAIT1
7BB7- AD 81 C0 3990 WAIT2      LDA $C081       ;CHECK UART STATUS
7BBA- 6A       4000            ROR A           ;(BIT 7)
7BBB- 90 FA    4010            BCC WAIT2
7BBD- 68       4020            PLA             ;RESTORE CHAR.
7BBE- 8D 82 Co 4030            STA $C082       ;SEND IT TO RS-232 OUTPUT
7BC1- 60       4040 EXIT       RTS
```

c. MAE Menu Listing (Applesoft BASIC)

```
20 HOME :D$ = CHR$ (84): PRINT TAB( 6)"MACRO ASSEMBLER/EDITOR (MAE)": PRINT
TAB( 12)"APPLE II VERSION"
30 PRINT : PRINT TAB( 13)"COPYRIGHT 1980"
40 PRINT TAB( 9)"EASTERN HOUSE SOFTWARE": PRINT : PRINT : PRINT : PRINT
: PRINT "ENTER CODE FOR DESIRED FUNCTION": PRINT
50 PRINT " A = ASSEMBLER/EDITOR (MAE)": PRINT " R = RELOCATING LOADER": PRINT
" W = MAE AND WORD PROCESSOR": PRINT " B = BASIC": PRINT " M = MONITOR"
60 PRINT : PRINT : PRINT : INPUT A$: IF A$ = 'A" THEN PRINT D$'BRUN MAE.EXE
": END
80 IF A$ = "R" THEN PRINT D$"BRUN RELOC.EXE": END
90 IF A$ = "B" THEN END
100 IF A$ = "M" THEN CALL - 151
102 IF A$ < > "W" THEN 20
105 PRINT D$"BLOAD MAE.EXE": FOR X = 1 TO 1000: NEXT X
117 PRINT D$"BLOAD WORDP.EXE": CALL 20480
```

17. Error Codes

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	
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 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 parameters 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.