# PET DISK MACRO
## ASSEMBLER/TEXT EDITOR
### (MAE)

<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Files Contained on the Diskette</td>
<td>2</td>
</tr>
<tr>
<td>3. Enhanced DOS Support Program</td>
<td>3</td>
</tr>
<tr>
<td>4. Text Editor (TED) Features</td>
<td>4</td>
</tr>
<tr>
<td>A. Commands</td>
<td>4</td>
</tr>
<tr>
<td>B. Entry/Deletion/Change of Text</td>
<td>11</td>
</tr>
<tr>
<td>5. Assembler (ASSM) Features</td>
<td>11</td>
</tr>
<tr>
<td>A. Source Statement Syntax</td>
<td>12</td>
</tr>
<tr>
<td>B. Label File (or Symbol Table)</td>
<td>21</td>
</tr>
<tr>
<td>C. Assembling</td>
<td>21</td>
</tr>
<tr>
<td>D. Creating a Relocatable Object File</td>
<td>22</td>
</tr>
<tr>
<td>E. MACROS</td>
<td>24</td>
</tr>
<tr>
<td>F. Conditional Assembly</td>
<td>26</td>
</tr>
<tr>
<td>G. Interactive Assembly</td>
<td>29</td>
</tr>
<tr>
<td>H. Default Parameters on entry to ASSM</td>
<td>29</td>
</tr>
<tr>
<td>6. Relocating the Relocating Loader</td>
<td>29</td>
</tr>
<tr>
<td>7. Error Codes</td>
<td>31</td>
</tr>
<tr>
<td>8. String Search and Replace Commands</td>
<td>32</td>
</tr>
<tr>
<td>A. EDIT Command</td>
<td>32</td>
</tr>
<tr>
<td>B. FIND Command</td>
<td>33</td>
</tr>
<tr>
<td>9. Control Codes (for Serial Device)</td>
<td>34</td>
</tr>
<tr>
<td>10. Connection of Serial Device</td>
<td>34</td>
</tr>
<tr>
<td>11. Examples</td>
<td>35</td>
</tr>
<tr>
<td>A. TED</td>
<td>35</td>
</tr>
<tr>
<td>B. ASSM</td>
<td>36</td>
</tr>
<tr>
<td>12. Getting Started with MAE</td>
<td>38</td>
</tr>
<tr>
<td>13. The MAE Simplified Text Processor (STP)</td>
<td>40</td>
</tr>
<tr>
<td>14. Special Notes</td>
<td>44</td>
</tr>
<tr>
<td>15. ASSM/TED Users Group</td>
<td>45</td>
</tr>
<tr>
<td>16. Example Listings</td>
<td>46</td>
</tr>
</tbody>
</table>

* * * *
* WRITTEN ENTIRELY IN MACHINE LANGUAGE BY *
* EASTERN HOUSE SOFTWARE, SERIAL #: *
* *
This manual and the object code (contained on floppy disk) is serial numbered and protected by a legitimate copyright. No part of this manual may be copied or reproduced without the express written permission of the copyright owner, Carl Moser. 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 who supplies information which leads to the prosecution of parties who violate this copyright.

We do not presume that you are or will violate copyright laws. Most users do not. Some though do, and may not realize the consequences for violation of this Federal Law. 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 can not be fully recovered if current illegal copying continues. Also, updates and program maintenance will have to be terminated if the return on investment is not sufficient.

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

If you encounter any problems, contact us at:

Eastern House Software
Carl Moser
3239 Linda Drive
Winston-Salem, N. C. 27106
1. INTRODUCTION

This Macro Assembler (ASSM) and Text Editor (TED) resides simultaneously in 10K bytes of memory (5000-77FF). The collective assembler and text editor is referred to as MAE. MAE was designed to work with the Commodore 32K PET (new ROMs) and the 2040 Disk Drive. Versions also exist for 40 column PETs retrofitted with 4.0 ROMs and for the CBM 80 column PETs (8032).

As mentioned, the MAE object code occupies 10K 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

These boundaries leave memory for the extended monitor ($1088-$17FF), the DOS support at upper memory, and memory for Basic and Machine Language programs ($0400-$1000).

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 pseudo op) and the .MC pseudo op.

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

Some unique features of MAE are:
- Coexists with PET Basic.
- Macro, Conditional Assembly, and Interactive Assembly.
- Labels up to 31 characters in length.
- Auto line numbering for ease of text entry.
- Creates both executable code in memory and relocatable object code on disk.
- Manuscript feature for composing letters and other text.
- Loading and Storing via Disk.
- Supports Serial I/O and/or IEEE printer.
- String search and replace capability, plus other powerful editing commands.
- Auto repeat of any key held down for 1/2 second.
- Capability to send command strings to 2040 Disk.

MAE uses a prompter character (3) to indicate that it is ready to accept commands. Command mnemonics referenced in this document are printed with the prompter (example JBR). When inputting a command, you should not type
"J" preceding the mnemonic.

Initial entry (or cold start) to MAE is at address $5000. If the break command (IBR) 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 PET Basic. This was accomplished by preserving Basics zero page variables. Thus, on cold start entry, MAE copies all 256 bytes of zero page to a save area ($7600-$76FF). On all exits (via IBR, JRU, JUS), 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.

Remember, MAE is a 10K system which uses memory from $5000-$77FF. You should protect MAE from Basic by setting the Basic variable HIMEM ($34, $35) to point to just below MAE and its text and label files. For example, to protect memory above $2000, enter $00 at location $34 and $20 at location $35.

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.

MAE is protected by a Copyright. This material may not be copied, reproduced, stored in a retrieval system, or otherwise duplicated without the written permission of the owner, Carl Moser. The purchaser may however make copies of the diskette for his own individual use for backup purpose.

The purchase of this software does not convey any license to manufacture, modify and/or copy this product in any manner.

2. FILES CONTAINED ON THE DISKETTE

The supplied diskette contains the following files:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAE/DOS.EXE</td>
<td>Enhanced DOS Support Program (Wedge)</td>
</tr>
<tr>
<td>EXTRAMON.EXE</td>
<td>Enhancement to PET M.L. Monitor and instructions - Courtesy Bill Seiler-CBM</td>
</tr>
<tr>
<td>EXTRAMON.INS</td>
<td>MAE object code</td>
</tr>
<tr>
<td>MAE.EXE</td>
<td>Relocating Loader object code</td>
</tr>
<tr>
<td>RELOC.EXE</td>
<td>Relocating Loader relocatable code</td>
</tr>
<tr>
<td>RELOC.REL</td>
<td>* Library of PET ROM locations</td>
</tr>
<tr>
<td>PET.LIB</td>
<td>* Some notes on MAE</td>
</tr>
<tr>
<td>MAE.NOT</td>
<td>Word Processor Program</td>
</tr>
<tr>
<td>WORDP.EXE</td>
<td>* Word Processor Instructions File and example of raw text</td>
</tr>
<tr>
<td>WORDP.INS</td>
<td>* File of some Machine Language Macros</td>
</tr>
<tr>
<td>MLMACROS.MLIB</td>
<td>* File of SWEET16 Macros (Use with PET16,</td>
</tr>
</tbody>
</table>
see #20 and #25 issues of Micro Magazine)
* IEEE Machine Language Driver Routines
* Example program which illustrates use
  of IEEE.LIB - displays disk sector

* = Source files in MAE format.

3. ENHANCED DOS SUPPORT PROGRAM

The first file on the MAE diskette is named MAE/DOS.EXE.
This file is an enhanced version of the DOS Support program supplied
by Commodore Business Machines. The standard DOS support commands
provided are:

- \ Load and run Basic program
- / Load Basic or Machine Language Program
> Read disk error channel
>*$n Display directory for drive n
>cmd Pass command string to disk drive

In addition to these commands, the MAE/DOS.EXE program provides
an auto-repeat key feature and the following MAE Support commands:

- AC Load MAE.EXE* (MAE Assembler/Text Editor) and begin
  execution at the cold start entry ($5000)
- AW Begin execution at the MAE warm start ($5003)
- MC Load EXTRAMON.EXE* (Extended Monitor) and begin
  execution at the cold start entry ($1000)
- MW Begin execution at the Extramon warm start entry ($0400)
- LC Load REL.EXE* (Relocating Loader) and begin execution
  at the cold start entry ($0500)
- LW Begin execution of Relocating Loader ($0500)

The purpose of these additional commands is to provide a quick and
convenient means to perform some of the more common MAE functions
but with minimal keystrokes. To illustrate this, compare the
following two methods of loading and executing the MAE.EXE* program:

1) Without MAE Support Commands:

```
>MAE.EXE*
SYS 20480
```

2) With MAE Support Commands:

```
AC
```

The difference is 20 keystrokes versus just 3!
4. 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 cursor-oriented screen editor plus the additional feature of automatic repeat of any key held down for 0.5 second.

When listing to the CRT or printer, the user has control of the output via the following keys:

- **STOP** - Temporarily halt outputting and await input of one of the following keys.
- **DEL** - Return to "1" level.
- **OFF** - Continue processing but suppress output except for errors.
- **Space** - Continue outputting after STOP.

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 JPR may be several non-space characters in length, MAE only considers the first two. For example, JPR, JPRI, JPRINT, and JPRETTY will be interpreted as the print command.

Some commands can be entered with various parameters. For example, JPRINT 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:

\[\text{Dn "drive:name"}\]

where:
- \(n\) is the device number (default = 8)
- drive is the disk drive number (0 or 1)
- name is the file name

Examples are:
- D9 "1:MAE.NOT"
- D8 "0:RELOC.REL"
- "DOS SUPPORT"
A description of each command follows:

JALPHA

Toggle shift character set from graphics to lower case and vice versa by toggling PET control at $E84C.

JASSEMBLE  file  w

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

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

JAUTO  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.

JBasic

Restore zero page and go to Basic.

JBREAK

Restore zero page and go to Monitor.

JCLEAR

Clear text file.
JCOPY 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.

JIC "command"

Pass disk commands to PET 2040 Disk. Any commands that can be entered with the PRINT# Basic statement may be entered.

Example: Output directory is JIC "$"
Scratch file TEST is JIC "$:TEST"

Note: Entry of JIC with no parameters results in display of error disk error channel messages.

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 8.

FIND t S1 t

String search.
See part 8.

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 listings.
3GET 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:

3GET "MEMTEST"
3GET "1:CRTDVR" 1000
3GET "0:UART" APPEND

3HARD 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. JHA SET turns this feature on, JHA CLEAR turns this feature off. x is the starting page number. JHA PAGE advances to top of next page.

Each time JHA 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.

JLABELS w

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

JMANUSCRIPT w

If w=SET, line numbers are not outputted when executing the JPR command. If w=CLEAR, line numbers are outputted when the JPR command is executed. Assembly output ignores the JMA command. If manuscript is to be generated using MAE, manuscript should be set and format clear (JMA SET, JFO 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 JNU 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 JSET command.

PASS file

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

PRINT x y

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

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.
JRUN 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.

JSET ts te ls le bs

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

JTI w n

Assign terminal input (keyboard) as PET if w=PET, 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 JTI SERIAL or JTI 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 PET.

JTO w n m

Assign terminal output (CRT or printer) as:
PET if w=PET, IEEE device #4 if w=IEEE, or serial
if w=SERIAL. If w=ALL, then output will be directed
to both the IEEE and the serial device.

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:

<table>
<thead>
<tr>
<th>n</th>
<th>baud rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>600</td>
</tr>
<tr>
<td>3</td>
<td>1200</td>
</tr>
<tr>
<td>4</td>
<td>2400</td>
</tr>
<tr>
<td>5</td>
<td>4800</td>
</tr>
<tr>
<td>6</td>
<td>7200</td>
</tr>
<tr>
<td>7</td>
<td>9600</td>
</tr>
</tbody>
</table>

JUSER

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

To delete a range of lines, use the JDE command. To edit an existing line or lines having similar characteristics, use the JED command.

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

To find a string, use the JFI command. To move or copy lines use the JMO or JCO 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 (JAU command). Any JAU x where x does not equal 0 puts the TEB in the auto line number mode on the next entry of a line number. To exit from this mode, type J//.

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 JFO command), the TEB will automatically line up the fields. Note: If a space is entered before the label, the TEB will line up the label in the next field. This should result in an assembler error when assembled. Commands, mnemonics, and pseudo ops may be entered as upper case or lower case characters. Labels in the program may be entered as upper or lower case characters but a label entered as upper case will be unique to the same label entered as lower case.

5. ASSEMBLER (ASSM) FEATURES

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

A third pass (via J0U), 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:

\[\begin{array}{cccc}
\text{line number} & \text{label} & \text{mnemonic} & \text{operand} & \text{comment} \\
\hline
\end{array}\]

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

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

The label is always entered immediately after the line number.

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

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

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

NOTE: It is 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 label with comment, then the label may be any length up to 79 characters regardless of the label length set with the JFORMAT command.
### TABLE A - 6502 Mnemonics

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

<table>
<thead>
<tr>
<th>AIC</th>
<th>CLD</th>
<th>LDA</th>
<th>SBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>CLI</td>
<td>LDX</td>
<td>SEC</td>
</tr>
<tr>
<td>ASL</td>
<td>CMP</td>
<td>LDY</td>
<td>SED</td>
</tr>
<tr>
<td>BCC</td>
<td>CPX</td>
<td>LSR</td>
<td>SEI</td>
</tr>
<tr>
<td>BCS</td>
<td>CPY</td>
<td>CLV</td>
<td>STA</td>
</tr>
<tr>
<td>BEQ</td>
<td>DEC</td>
<td>ORA</td>
<td>STX</td>
</tr>
<tr>
<td>BIT</td>
<td>DEX</td>
<td>PHA</td>
<td>STY</td>
</tr>
<tr>
<td>BMI</td>
<td>DEY</td>
<td>PHP</td>
<td>NOP</td>
</tr>
<tr>
<td>BNE</td>
<td>EOR</td>
<td>PLA</td>
<td>TAX</td>
</tr>
<tr>
<td>BPL</td>
<td>INC</td>
<td>PLP</td>
<td>TAY</td>
</tr>
<tr>
<td>BRK</td>
<td>INX</td>
<td>ROL</td>
<td>TXS</td>
</tr>
<tr>
<td>BVC</td>
<td>INY</td>
<td>ROR</td>
<td>TXA</td>
</tr>
<tr>
<td>BVS</td>
<td>JMP</td>
<td>RTI</td>
<td>TXS</td>
</tr>
<tr>
<td>CLC</td>
<td>JSR</td>
<td>RTS</td>
<td>TYA</td>
</tr>
</tbody>
</table>
TABLE B - Pseudo Ops

.BA  label exp.

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

.BY

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

.CE

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

.CT

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

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

label .DE  label exp.

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

.DE label expr.
Define a block of storage. For example, if label expr. equated to 4, then ASSM will skip over 4 bytes.
Note: The initial contents of the block of storage is undefined.

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

.EJ
Eject to top of next page if JCA SET was previously entered.

.EN
Indicates the end of the source program.

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

.FI 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

Output 7 followed with space and then accept exactly 4 hex digits. These hex digits will be assigned to label and stored in the label file.

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

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.

.LC

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

.LS

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

.MC label exp.

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

.MD

Macro definition. See part 5E.
.ME

Macro end of definition. See part 5E.

.MG

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

.OC

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

.OS

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

.PR "text"

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

.RC

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

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

.SE label exp.

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

.SI label exp.

Store the address calculated from the label expression in the next two memory locations. Consider this address as being an internal address.
TABLE C - Label Expressions

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

Symbolic Labels:
One to 31 characters consisting of the ASCII characters as previously defined. The maximum number of characters is set by the JFORMAT 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 precedes 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:
+ addition
- subtraction

Examples of some valid label expressions follow:
LDA #$1101 ;LOAD IMMEDIATE $0D
STA *TEMP+$01 ;STORE AT BYTE FOLLOWING TEMP
LDA $471E36 ;LOAD FROM LOCATION $1E36
JMP LOOP+C-$461 ;JMP TO CALCULATED ADDRESS
BNE =+8 ;BRANCH TO CURRENT PC PLUS 8 BYTES

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

ASL A+8 does not result in accumulator addressing but instead references a memory location.
TABLE D - Addressing Mode Formats

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

Absolute:
  LDA label exp.

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

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

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

Indexed Indirect:
  LDA (label exp.,X)

Indirect Indexed:
  LDA (label exp.),Y

Indirect:
  JMP (label exp.)

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

Implied:
  TAX ;OPERAND FIELD IGNORED
  CLC

Relative:
  BEQ label exp.
B. Label File (or Symbol Table)

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

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

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

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

C. Assembling

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

These modules can be linked together during assembly via a control file. If used, the control file must be the first file to be assembled. This file must be in the text buffer when the JAS command is issued, or its name must be specified in the JAS command (example: JAS "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 D8 "X.M01"
.FI D8 "Y.M02"
.FI D8 "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 PASS!. If for some reason you terminate the assembly on the second pass, you may restart at the beginning of the second pass using the JPASS command.

D. Creating a relocatable object file (JOE)

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 .BE pseudo-op. Non-symbolic labels (example: $0169) are also considered as being external (or fixed). All other labels (including those defined via the .DI pseudo-op) are considered as internal. Addresses associated with internal labels can be altered by an offset when the program is loaded via the relocating loader.

Also, the .SE stores a two byte external address and the .SI stores a two byte internal address. 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 PET ROM routines or any location whose address remains the same no matter where the program is located. Expressions consisting of internal and external labels will be combined and considered an internal address. A label expression consisting entirely of external labels will be combined and considered as external.

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

Start assigning zero page references at location $0000, absolute variable locations at $0400, and begin assembly of the program at $2000. Next assemble the program via JAS, and then issue the JOOUT 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 $500 if in the monitor or SYS 1280 if in Basic. The relocating loader will request the following:

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

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

3) ABS OFFSET? Address to begin assignment of absolute internal
When the file has been relocated in memory, it can be saved on disk (using Extramon) 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 $8060$, and the absolute variables at $4000$. We would respond to the relocating loader as follows:

FILENAME? Do "1: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 return to exit the Relocating Loader
E. Macros

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

```
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 code for the associated variable in the call for expansion.

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

There is a way to get 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 listings. 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 6582 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 88 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.

IFP label exp.

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

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

***

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

SET label=label exp.

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

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

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

CHAR.LINE .DE 40

.*

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

.*

***

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

.*

***

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

.*

***

;COMMON CODE FOR ALL

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

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

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

;SORT CODE FOLLOWS
SORT.CALL
.

RTS

...ABC SET EXPAND=1
***
.ME
```

In this example, EXPAND is initially set to 0. When the macro is expanded for the first time, EXPAND equals 0 and the code at SORT.CALL will be assembled along with a JSR to and a JMP around the sort subroutine. Also, the first expansion sets EXPAND to 1. On each 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.

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

**NOTE:** Never specify a label as the operand in the `.IN` pseudo op that has been defined by the `.DE`, `.DI`, or SET pseudo ops. The reason is that these pseudo ops initialize the address assigned to associated labels on both assembly passes while all other labels are initialized only on the first pass. Since the `.IN` pseudo op accepts input on the first pass only, usage of labels defined by `.DE`, `.DI`, and SET will cause different label values on pass 1 versus pass 2.

H. Default Parameters on entry to ASSM

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

6. 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 PET monitor. Just remember, execution begins at the address specified for the PGM EXE OFFSET - not $0500 as for the executable copy supplied (RELOC.EXE).
7. 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:

<table>
<thead>
<tr>
<th>ERROR CODE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B</td>
<td>.EN in non .CT file when .CT file exists.</td>
</tr>
<tr>
<td>1A</td>
<td>.EN missing in .CT designated file.</td>
</tr>
<tr>
<td>19</td>
<td>Found .FI in non .CT file.</td>
</tr>
<tr>
<td>18</td>
<td>Checksum error on disk load.</td>
</tr>
<tr>
<td>17</td>
<td>Syntax error in JED command.</td>
</tr>
<tr>
<td>16</td>
<td>Device numbers 0,1,2,3 not allowed.</td>
</tr>
<tr>
<td>15</td>
<td>Multiple .CT assignment.</td>
</tr>
<tr>
<td>14</td>
<td>Command syntax error or out of range error.</td>
</tr>
<tr>
<td>13</td>
<td>Missing parameter in JNU command.</td>
</tr>
<tr>
<td>12</td>
<td>Overflow in line # renumbering.</td>
</tr>
<tr>
<td>11</td>
<td>CAUTION: You should properly renumber the text file for proper command operation.</td>
</tr>
<tr>
<td>10</td>
<td>Overflow in text file - line not inserted.</td>
</tr>
<tr>
<td>0F</td>
<td>Overflow in label file - label not inserted.</td>
</tr>
<tr>
<td>0E</td>
<td>MAE expected hex characters, found none.</td>
</tr>
<tr>
<td>0D</td>
<td>Illegal character in label.</td>
</tr>
<tr>
<td>0C</td>
<td>Unimplemented addressing mode.</td>
</tr>
<tr>
<td>0B</td>
<td>Error in or no operand.</td>
</tr>
<tr>
<td>0A</td>
<td>Found illegal character in decimal string.</td>
</tr>
<tr>
<td>09</td>
<td>Undefined label (may be illegal label).</td>
</tr>
<tr>
<td>08</td>
<td>.EN pseudo op missing.</td>
</tr>
<tr>
<td>07</td>
<td>Duplicate label.</td>
</tr>
<tr>
<td>06</td>
<td>Label missing in .DE or .DI pseudo op.</td>
</tr>
<tr>
<td>05</td>
<td>.BA or .MC operand undefined.</td>
</tr>
<tr>
<td>04</td>
<td>Illegal pseudo op.</td>
</tr>
<tr>
<td>03</td>
<td>Illegal mnemonic or undefined macro.</td>
</tr>
<tr>
<td>02</td>
<td>Branch out of range.</td>
</tr>
<tr>
<td>01</td>
<td>Not a zero page address.</td>
</tr>
<tr>
<td>00</td>
<td>Error in command input.</td>
</tr>
<tr>
<td>ED</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>ERROR CODE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>2F</td>
<td>Overflow in file sequence count (2**16 max.)</td>
</tr>
<tr>
<td>2E</td>
<td>Overflow in number of macros (2**16 max.)</td>
</tr>
<tr>
<td>2D</td>
<td></td>
</tr>
<tr>
<td>2C</td>
<td></td>
</tr>
</tbody>
</table>
8. STRING SEARCH AND REPLACE COMMANDS

A. Edit Command

A powerful string search and replace, and line edit capability is provided via the JEDIT 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

JEDIT ts1+s2+t %d * x y

Where:

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

Asterisk (*) prompter subcommands:

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

Defaults:

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

If no * or # entered then print all lines altered.
For example, to replace all occurrences of the label LOOP with the label START between lines 100 and 600, enter:

```
JEDIT /LOOP/START/ 100 600
```

To simply delete all occurrences of LOOP, enter:

```
JEDIT /LOOP// 100 600
```

You may use the * and # as described above.

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

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

Form 2

```
JEDIT n
```

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

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

B. Find Command

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

```
JFIND tS1t # x y
```

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

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

At the end of the JFIND 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: JFIND /%/#
9. CONTROL CODES (Serial Device)

The following applies to the optional serial device connected to the PET. 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 \cC, and 18 as \cX, etc.

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

Control codes which have special functions are:

<table>
<thead>
<tr>
<th>CODE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Null (hex 00)</td>
</tr>
<tr>
<td>B</td>
<td>Restore zero page and go to Basic</td>
</tr>
<tr>
<td>C</td>
<td>Restore zero page and go to Monitor</td>
</tr>
<tr>
<td>f</td>
<td>Bell</td>
</tr>
<tr>
<td>H</td>
<td>Backspace (delete previously entered char.)</td>
</tr>
<tr>
<td>I</td>
<td>Horizontal tab to next 8-th char. position</td>
</tr>
<tr>
<td>J</td>
<td>Line feed</td>
</tr>
<tr>
<td>M</td>
<td>Carriage return</td>
</tr>
<tr>
<td>T0</td>
<td>Continue processing but no output (same as DEL)</td>
</tr>
<tr>
<td>T1</td>
<td>Continue after stop via break key</td>
</tr>
<tr>
<td>T2</td>
<td>Delete entire line altered</td>
</tr>
<tr>
<td>T3</td>
<td>Restore zero page and jump to location $0000. (you may reenter at $5003)</td>
</tr>
<tr>
<td>T7</td>
<td>Terminate processing and go to &quot;1&quot; level</td>
</tr>
<tr>
<td>T7</td>
<td>Escape character</td>
</tr>
</tbody>
</table>

* = Non-printing control character.

10. CONNECTION OF A SERIAL DEVICE

A serial device may be connected to your PET and controlled by MAE software. MAE generates data in TTY (or RS232) data format on the USER port (bit 7 pin L = output, bit 6 pin K = input). The data format consists of one start, seven data, and two stop bits. Since these signals on the user port are TTL levels, circuitry may be required to provide a proper electrical interface. We have found, though, that RS232 terminals such as the Synertek KTM-80 can be connected directly to the user port. If you do provide interface circuitry, you should not invert the signals as they are in positive true state.

The commands JTI and JTO are provided to direct MAE to input or output on this serial port.
11. EXAMPLES

A. TED Examples

#1 Illustrate two ways to load MAE using MAE/DOS support and begin execution at cold start.
   /MAE.EXE* or AC
   SYS 20480

#2 Illustrate two ways to load and initialize the extended monitor (EXTRAMON).
   /EXTRAMON.EXE or MC
   SYS 4096

#3 Illustrate entry of text.
   JAUTO 10
   1000;THIS IS A TEST
   1010 LOOP LDA VALUE,Y
   1020 NOP
   1030 END.PGM .EN
   1040// (Note, enter // to exit auto line -ing

#4 Illustrate listing of text.
   JPRINT
   1000 ;THIS IS A TEST
   1010 LOOP LDA VALUE,Y
   1020 NOP
   1030 END.PGM .EN

   //

#5 Put file to disk (device 9, drive 0) with name TEST.
   JPUT D9 "0:TEST"

#6 Get file from disk (device 9, drive 1) named TEST.
   JGET "0:TEST"

   Note: The default is device 8.

#7 Assemble file CRTDVR and generate a listing.
   JASSM "CRTDVR" LIST

#8 Output directory for drive 0.
   JDC "#$0"

#9 Scratch file TEST.
   JDC "S:TEST"

#10 Read disk error channel.
   JDC

#11 Direct output to IEEE printer (device #4).
   JTO IEEE

#12 Direct output to Serial device at 300 baud with 10 pad bits.
JTO SERIAL 1 10

#13 Assign serial device as input (keyboard) and output (with 2 read bits).
    JTI TERMINAL 2

S  → user types S on serial keyboard then types RETURN.

#14 Find all occurrences of the text LDA.
    JFIND /LDA/

#15 Replace all occurrences of LDA FA with LDA *FA between lines 1000 and 2000.
    JEDIT /LDA FA/LDA *FA/ 1000 2000

#16 Provide for 15 characters per label.
    JFORMAT SET 15

#17 Output all fixed (external) labels.
    JLABELS FIXED

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

#19 Move lines 100 thru 200 to after line 9000.
    JMOVE 9000 100 200

#20 Print lines 900 thru 976
    JPRINT 900 976

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

#22 Go to Basic.
    JBASIC
        return via SYS 20480 (cold start)
        or SYS 20483 (warm start)

#23 Go to Machine Language Monitor.
    JBREAK
        return via G 5000 (cold start)
        or G 5003 (warm start)

#24 Run assembly program at symbolic label BOX.
    JRUN BOX

#25 Change from Uppercase/Graphics character set to Lowercase/Uppercase or vice versa.
    JALPHA

B. ASSM Examples

#1 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.
   CRT .DE $FFD2

#4 Assign an internal work location in zero page.
   WORK .DI $0

#5 Allocate 6 bytes of storage.
   TABLE .DS 6

#6 Define label EOI as mask with bit 6 set and show use in AND statement.
   EOI .DE $01000000
   AND #EOI

#7 Load the low address part of the label VALUES in register X and high part in register Y.
   LDX #L,VALUES
   LDY #H,VALUES

#8 Give example of .BY pseudo op.
   .BY 'ALARM CONDITION ON MOTOR 1' $0D $0A

#9 Store the address of the internal label TABLE and the external label PETOUT.
   .SI TABLE
   .SE PETOUT

#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 3SET 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 1FO command) are allowed if defined on a line with no mnemonics.

#12 Reference the call to the PET RDT ascii character routine so the relocating loader will not alter the address during loading.
   RDT .DE $FFCF
12. GETTING STARTED WITH MAE

An extended monitor program (EXTRAMON) is contained on the supplied diskette. This program, developed by Bill Seiler of CBM, provides many additional monitor commands which will be of tremendous help in your program development at the object code level. Therefore, we recommend that you load EXTRAMON with MAE. If you are unfamiliar with EXTRAMON, load the Basic program EXTRAMON.INS for an interactive review of its many powerful commands.

Load MAE/DOS, MAE, and EXTRAMON as follows:

1-) Insert supplied diskette in disk drive 0
2-) Load MAE/DOS support program via LOAD "*",8
3-) Type RUN to initialize DOS support program.
   Note that the screen provides a description of 6 additional DOS commands: AC, AW, MC, MW, LC, LW.
   These commands are aids to quickly load and transfer control to other MAE programs.
4-) Type MC to load and initialize the extended monitor.
5-) Type .X to return to BASIC.
6-) Type AC to load and cold start the MAE Assembler/Text Editor.

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 JSET command can be used to display this range again or alter the file boundaries.

Remember, to exit MAE, issue either the JBASIC or JBREAK commands to go to Basic or to Monitor. You may reenter MAE via $5003 (warm start – everything preserved) or $5000 (cold start – everything cleared to default state). If you are in BASIC, type AW to warm start MAE. This is the same as .G 5003 and SYS 20483.

Also, you should note that EXTRAMON occupies memory at $1000-$17FF and MAE occupies $5000-$77FF. EXTRAMON also sets the Basic variable HIMEM to $1000 to indicate the end of memory available for Basics use. This in effect protects EXTRAMON, MAE, and other programs above $1000 from being "clobbered" by Basic. You will want to manually reset HIMEM if Basic issues an out of memory error. HIMEM may be reset to its cold start value as follows:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0034</td>
<td>$00</td>
</tr>
<tr>
<td>$0035</td>
<td>$80</td>
</tr>
</tbody>
</table>

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

JGET "MAE.NOT"
JFORMAT CLEAR   — turn formatting off
JAL              — enter upper/lower case mode

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 4, assembler features in part 5, and then the examples in part 10.

We hope you find MAE to be an excellent program development aid and a worthwhile investment. Happy Assembling!!!
13. MAE Simplified Text Processor (STP)

The MAE Simplified Text Processor (STP) is a word processor program designed specifically to work with the MAE text editor. The primary purpose of this word processor was to provide a simplified means to process program documentation and for other text processing needs. This simplicity was accomplished with a set of 16 easily remembered word processing functions, and usage of an already familiar text editor to enter and edit the raw text.

STP, unlike the CBM Word Pro 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.

To instruct the word processor to perform a word processing function, one inserts text macros in the text to be formatted. A text macro always begins with a period (.), always begins in column 1, may be entered as upper or lower case, and may or may not have associated parameters. The following are the macros provided by the STP word processor:

VERTICAL SPACING (.vspace n)

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

TEMPORARY INDENT (.sn)

To indent n spaces on the next line, use the .sn macro where 

n = the number of spaces to indent. For example, .s5 will indent the next line 5 spaces from the right.

MARGIN CONTROL (.m n p α 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)
α = 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 α = 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 \( \alpha \) 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** (\texttt{.nofill})

The \texttt{.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, words are not combined to fill to the specified margins.

**BEGIN A NEW PAGE** (\texttt{.ff})

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

**LITERAL SPACE** (\texttt{↑ 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 (\texttt{↑}) characters may be inserted into the text. The caret will not be printed when the text is processed but instead a space will be printed for each occurrence of the caret.

**TURN ON JUSTIFICATION** (\texttt{.ju})

The \texttt{.ju} macro may be entered in order to restore justification. \texttt{.ju} is normally used to revert back to justification after using the \texttt{.nofill} macro.

**RAGGED RIGHT MARGIN** (\texttt{.rr})

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

**RAGGED LEFT MARGIN** (\texttt{.rl})

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

**SKIP NEXT N LINES** (\texttt{.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 \texttt{.12}. If you enter \texttt{.1} by itself, one will be
assumed. Thus \texttt{.1} and \texttt{.11} are equivalent and each will result
in a movement to the next line.

\textbf{CENTER LINE OF TEXT (.c text)}

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

\textbf{SWAP JUSTIFICATION MODES (.swap)}

This macro is used to switch from \texttt{.rr} mode to \texttt{.rl} and
vice versa.

\textbf{PARAGRAPH SPECIFICATION (.p d r) and PARAGRAPH IDENTIFICATION (.p)}

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

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

\textbf{PAGE TITLE (.t# title text)}

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

\textbf{PAGE FOOTERS (.foot# foot text)}

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

\textbf{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 \texttt{.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 \texttt{.shape 1} at the beginning of the
text file.
The .share 2 macro may be used to create a user defined share. In order to define the desired share, .set macros are used to make entries in the user share table corresponding to the desired share. 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 share table. But it should be noted that .set 0 l 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 \( l \) and print width of \( p \). This is useful as you can set all lines in the user share table to a particular margin and then use non 0 values to change certain lines to form the desired share.

Note: Always enter the .share 2 macro before the .set macros. The reason is that as soon as the .share 2 macro is encountered, it fills the user share 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 .share 2 defaults of 0 and 40.

If .share 2 is entered and no share 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 PETs which have 40 column screens.

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

HOW TO USE THE STP WORD PROCESSOR

1) Load the word processor and MAE via:
Note: Basics Hi Mem pointer ($34, $35) is set above the word processor. Either poke this pointer to $0700 or refrain from using Basic commands which use program memory space.

2) Enter upper case/lower case mode via the JAL command. Clear format mode via JFORMAT 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:

   IRUN $700  ← for output to CRT only
   IRUN $703  ← for output to CRT and Printer

EXAMPLE

A raw text file named WORDP.INS is contained on the diskette. Type 1GET "WORDP.EXE" to load this file. Type 1PRINT to examine the raw text with associated macros. Type 1RUN $700 (CRT only) or 1RUN $703 (CRT and Printer) to execute the word processor and output the text in word processor format. Note: If you output this to a 40 column PET, 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.

14. SPECIAL NOTES

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

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

* Immediately after using the PET Machine Language Monitor to save a program, always X to Basic and then SYS 1024 back to the monitor. The reason is that the IRQ vectors are destroyed by the PET save software.

* If you are going to use MAE and Basic together, alter HIMEM ($0034, $0035) 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.

* Due to some "strange" disk problems, never use the save with replacement feature (@) - Example: JPUT "@1:TEST.M01" or even SAVE "@:FILE",8.

* 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

15. ASSM/TED USERS GROUP

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 MAE compatible. The cost per diskette is also minimal but the information is extremely useful.

Some of the more notable programs on the first diskette are:

  UNASSEMBLER/MAE - Basic program which disassembles into a disk file compatible with MAE.
  KEYSORT - M.L. program for sorting Basic variables
  MAE/DOS.ASM - Source for the MAE/DOS Support program
  EPROM PROGMR - 2716/2732 EPROM programmer which connects to User Port.
  PET16 - Sweet 16 interpreter adapted for the PET

For more details, contact:
16. EXAMPLE LISTING

An example of a program in MAE's syntax follows. This program is the UART driver contained on the supplied diskette under files: UART.CTL, UART.M01, UART.M02, 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 renumeration.
F'HGE 47

2000

2010 ; DESIGNATE AS CONTROL FILE
2020 ; CONTINUE IF ERRORS
2030 .BA $2000
2040 
2050 
2060 ; +++++ DEFINITIONS ++++++
2070 
2080 0090 PIA.PORT .DE #E841 ; PIA DATA PORT
0100 PIA.DIR .DE #E843 ; PIA DIRECTION PORT
0110 0120 MSK.IN .DE #E201000000 ; INPUT IS ON BIT 6
0130 MSK.OUT .DE #E10000000 ; OUTPUT IS ON BIT 7
0140 0150 0160 ; UART CONTROL PARAMETERS:
0170 ;-------------------------------------
2010 0180 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 ;
0240 ;
0250 ;
0260 ;
0270 ;
0280 ;
0290 ;
0300 ;

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 #$FF-MSK.IN
2009- 09 80 0080 ORA #MSK.OUT
200B- 8D 43 E8 0090 STA PIA.DIR
200E- 20 47 20 0100 JSR GET.BIT
2011- D0 FB 0110 LP1 BNE LP1 ; BR. IF ALREADY SPACING
2013- AD 41 E8 0120 LP2 LDA PIA.PORT
2016- 29 40 0130 LP3 AND #MSK.IN
2018- F0 F9 0140 BEQ LP2 ; BR. IF MARKING
201A- A0 00 0150 LDY #00 ; CLEAR FOR DELAY FACTOR
201C- AD 41 E8 0160 LP3 LDA PIA.PORT ; GET BIT
201F- 29 40 0170 LP4 AND #MSK.IN
2021- F0 07 0180 BEQ GOT.COUNT
2023- C0 FF 0190 CPY #$FF
2025- F0 F5 0200 BEQ LP3

.Page 47

07F6 233E-2B34 UART.M01

0010 ; +++++ SET BAUD RATE +++++
0020 ;
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 #$FF-MSK.IN
2009- 09 80 0080 ORA #MSK.OUT
200B- 8D 43 E8 0090 STA PIA.DIR
200E- 20 47 20 0100 JSR GET.BIT
2011- D0 FB 0110 LP1 BNE LP1 ; BR. IF ALREADY SPACING
2013- AD 41 E8 0120 LP2 LDA PIA.PORT
2016- 29 40 0130 LP3 AND #MSK.IN
2018- F0 F9 0140 BEQ LP2 ; BR. IF MARKING
201A- A0 00 0150 LDY #00 ; CLEAR FOR DELAY FACTOR
201C- AD 41 E8 0160 LP3 LDA PIA.PORT ; GET BIT
201F- 29 40 0170 LP4 AND #MSK.IN
2021- F0 07 0180 BEQ GOT.COUNT
2023- C0 FF 0190 CPY #$FF
2025- F0 F5 0200 BEQ LP3

07F6 233E-2B34 UART.M01

0010 ; +++++ SET BAUD RATE +++++
0020 ;
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 #$FF-MSK.IN
2009- 09 80 0080 ORA #MSK.OUT
200B- 8D 43 E8 0090 STA PIA.DIR
200E- 20 47 20 0100 JSR GET.BIT
2011- D0 FB 0110 LP1 BNE LP1 ; BR. IF ALREADY SPACING
2013- AD 41 E8 0120 LP2 LDA PIA.PORT
2016- 29 40 0130 LP3 AND #MSK.IN
2018- F0 F9 0140 BEQ LP2 ; BR. IF MARKING
201A- A0 00 0150 LDY #00 ; CLEAR FOR DELAY FACTOR
201C- AD 41 E8 0160 LP3 LDA PIA.PORT ; GET BIT
201F- 29 40 0170 LP4 AND #MSK.IN
2021- F0 07 0180 BEQ GOT.COUNT
2023- C0 FF 0190 CPY #$FF
2025- F0 F5 0200 BEQ LP3

07F6 233E-2B34 UART.M01

0010 ; +++++ SET BAUD RATE +++++
0020 ;
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 #$FF-MSK.IN
2009- 09 80 0080 ORA #MSK.OUT
200B- 8D 43 E8 0090 STA PIA.DIR
200E- 20 47 20 0100 JSR GET.BIT
2011- D0 FB 0110 LP1 BNE LP1 ; BR. IF ALREADY SPACING
2013- AD 41 E8 0120 LP2 LDA PIA.PORT
2016- 29 40 0130 LP3 AND #MSK.IN
2018- F0 F9 0140 BEQ LP2 ; BR. IF MARKING
201A- A0 00 0150 LDY #00 ; CLEAR FOR DELAY FACTOR
201C- AD 41 E8 0160 LP3 LDA PIA.PORT ; GET BIT
201F- 29 40 0170 LP4 AND #MSK.IN
2021- F0 07 0180 BEQ GOT.COUNT
2023- C0 FF 0190 CPY #$FF
2025- F0 F5 0200 BEQ LP3

07F6 233E-2B34 UART.M01

0010 ; +++++ SET BAUD RATE +++++
0020 ;
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 #$FF-MSK.IN
2009- 09 80 0080 ORA #MSK.OUT
200B- 8D 43 E8 0090 STA PIA.DIR
200E- 20 47 20 0100 JSR GET.BIT
2011- D0 FB 0110 LP1 BNE LP1 ; BR. IF ALREADY SPACING
2013- AD 41 E8 0120 LP2 LDA PIA.PORT
2016- 29 40 0130 LP3 AND #MSK.IN
2018- F0 F9 0140 BEQ LP2 ; BR. IF MARKING
201A- A0 00 0150 LDY #00 ; CLEAR FOR DELAY FACTOR
201C- AD 41 E8 0160 LP3 LDA PIA.PORT ; GET BIT
201F- 29 40 0170 LP4 AND #MSK.IN
2021- F0 07 0180 BEQ GOT.COUNT
2023- C0 FF 0190 CPY #$FF
2025- F0 F5 0200 BEQ LP3
2027- C8 0220  INY
2028- D0 F2 0230 SKP_FF BNE LP3
202A- 98 0250 GOT.COUNT TYA ;MOVE COUNT TO R(A)
202B- A0 00 0260 LIY #00
202D- D3 3F 20 0270 LP.FI CMP TBLBAUD,Y
2030- B0 03 0280 BCS GOTBAUD
2032- C8 0290 INY
2033- D0 F8 0300 BNE LP.FI
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
203E- FF 0360
2040- 93 0370
2041- 4A 0380 TBLBAUD .BY 255 ; >= 110
2042- 25 0390 .BY 147 ; >= 300
2043- 12 0400 .BY 74 ; >= 600
2044- 09 0410 .BY 37 ; >= 1200
2045- 07 0420 .BY 18 ; >= 2400
2046- 00 0430 .BY 10 ; >= 4800
2047- AD 41 E8 0440 .BY 7 ; >= 7200
2048- 29 40 0450 .BY 0 ; >= 9600
204C- 60 0460
204D- AD 01 20 0500 0510
2050- 18 0520 DEL0.5 LDA BIT.TIME
2051- D8 0530 CLC
2052- 69 08 0540 CLD
2054- A8 0550 ADC #08
2055- 4C 5B 20 0560 TAY
2055- 4C 5B 20 0570 JMP EN0.5
2058- AC 01 20 0580 BNE LOOPDEL1
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
2066- 00 0680 ;---DELAY=5X+19
2067- 00 0690 UD.TBL1 .BY 00 ;DELAY FULL FOR 110 BAUD
2068- 00 0700 .BY 00 ;DELAY FULL FOR 300 *
2069- 00 0710 .BY 00 ;DELAY FULL FOR 600 *
206A- 99 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 08 ;DELAY FULL FOR 9600 *
206E- 00 0770 0780 ;---DELAY=5X+28
206E- 00 0790 .BY 00 ;DELAY 0.5 FOR 110 BAUD
206F- 00 0800 .BY 00 ;DELAY 0.5 FOR 300 *
2070- 00 0810 .BY 00 ;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 10 ;DELAY 0.5 FOR 4800 *
2074- 03 0850 .BY 03 ;DELAY 0.5 FOR 7200 *
2075- 01 0860 .BY 01 ;DELAY 0.5 FOR 9600 *

2076- B9 8C 20 0860 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 0900 LOOPDEL2
2080- 48 0940 .BY $68 $68 $68 $68 $68 $68 $68
2081- 68 68 68 0950 .BY $68 $68 $68 $68 $68 $68 $68
2084- 68 68 68 0960 ;---DELAY=54X+22
2087- 68 0970 UD.TBL2 .BY 167 ;DELAY FULL FOR 110 BAUD
2088- 88 0980 .BY 60 ;DELAY FULL FOR 300 *
2089- D0 EF 0990 .BY 30 ;DELAY FULL FOR 600 *
208B- 60 0940 RTS
208E- 1E 0940
208F- 00 00 00 1000 .BY 00 00 00 00 00
2092- 00 00 1010
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 *
2097- 08 0040 UART.OUT PHP
2098- 78 0050 SEI
2099- 20 9E 20 0060 JSR UART.OUT1 ;RESTORE PSR AND RETURN
209C- 28 0070 PLP
209D- 60 0080 RTS
209E- 48 0090
209F- 49 FF 0100 UART.OUT1 PHA ;SAVE CHAR.
20A1- 48 0110 EOR ###FF ;INVERT
20A2- A2 0B 0120 PHA
20A4- 38 0130 LDX #11 ;11 BITS: 1 STOP, 8 DATA
20A5- 20 D2 20 0140 SEC
20A6- 20 58 20 0150 JP BIT.OUT ;BIT TO PORT
20A8- 20 58 20 0160 JSR DLVFULL ;DELAY FULL BIT TIME
20AB- 68 0170 PLA ;RESTORE $A
20AC- 4A 0180 LSR A ;NEXT BIT
20AD- 48 0190 PHA ;AND SAVE
20AE- CH 0200 DEX
20AF- D0 F4 0210 BNE LP.OUT ;LOOP
20B1- 68 0220 PLA ;REMOVE JUNK
<table>
<thead>
<tr>
<th>Address</th>
<th>Hex</th>
<th>Machine Code</th>
<th>Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0B2-68</td>
<td>0230</td>
<td>PLA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B3-29</td>
<td>0240</td>
<td>AND #02F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B5-0D</td>
<td>0250</td>
<td>CMP #00D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B7-0B</td>
<td>0260</td>
<td>BEQ PAD.DEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B9-0A</td>
<td>0270</td>
<td>CMP #01D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0BB-07</td>
<td>0280</td>
<td>BEQ PAD.DEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0BD-60</td>
<td>0290</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0BE-48</td>
<td>0310</td>
<td>PAD.DELX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0BF-00</td>
<td>0320</td>
<td>CPX #00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C1-0C</td>
<td>0330</td>
<td>JMP PAD.DELEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C4-48</td>
<td>0350</td>
<td>PAD.DEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C5-00</td>
<td>0360</td>
<td>LDX #00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C8-06</td>
<td>0370</td>
<td>PAD.DELLEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CA-20</td>
<td>0380</td>
<td>LP.PDEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CD-CA</td>
<td>0390</td>
<td>DEX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0CE-FA</td>
<td>0400</td>
<td>BNE LP.PDEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0DE-68</td>
<td>0410</td>
<td>EX.DEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0DF-60</td>
<td>0420</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2002-AD 41 E8 0440 BIT.OUT  LDA PIA.PORT ;PUT BIT
2005-29 7F 0450 AND #FF-MSK.OUT
2008-90 02 0460 BCC SKP.BOUT
2009-09 80 0470 ORA #MSK.OUT
20DB-8D 41 E8 0480 SKP.BOUT  STA PIA.PORT
20DE-60 0490 RTS

0320

.FI D8 "UART.M03" ;UART INPUT DRIVER

0248 233E-2586 UART.M03

0100 ;       ++++ UART INPUT ++++
0200
0300

20DF-08 0040 UART.IN  PHP
20E0-78 0050 SEI
20E1-09 00 0060 LDA #00  ;CLEAR CHAR.
20E3-48 0070 PHA ;*
20E4-20 47 20 0080 LP.UI1  JSR GET.BIT  ;GET BIT
20E7-DF FB 0090 BNE LP.UI1  ;LOOP UNTIL NO BIT
20E9-20 47 20 0110 LP.UI2  JSR GET.BIT  ;GET BIT
20EC-FD FB 0120 BEQ LP.UI2  ;LOOP UNTIL START BIT
20EE-20 4D 20 0140 JSR DEL0.5  ;DELAY UNTIL MIDDLE OF START BIT
20F1-20 47 20 0150 LP.UI3  JSR GET.BIT  ;GET BIT
20F4-38 0160 SEC  ;ASSUME SPACE
20F5-0D 01 0170 BNE SKP.UI1  ;NO IT IS MARK
20F7-18 0180 CLC  ;ROTATE RIGHT INTO CARRY
20FB-68 0190 SKP.UI1  PLA
20FD-68 0200 ROR A
20FA-00 07 0210 BCS DONE.UI
20FC-48 0220 PHA
20FD-58 20 0230 JSR DLYFULL  ;DELAY UNTIL MIDDLE OF NEXT
PAGE 51

2101- 90 EE 0250  BCC LP.UI3  GLOOP FOR NEXT BIT
2103- 49 FF 0260 DONE.UI  EOR #FF  ;INVERT
2105- 29 7F 0270  AND #7F  ;CLEAR BIT 7
2107- 28 0280  PLP  ;RESTORE PSR AND RETURN
2108- 60 0290  RTS
0330
0340
0350
0360 END.PGM .EN

;INITIALIZE PSS1

--- LABEL.FILE: ---

BIT.OUT =20D2
LILYFULL =2058
END.PGM =2109
GOT.COUNT =202A
LOOPDEL2 =207A
LP.UI1 =20E4
LP.UOUT =20A5
LP3 =201C
NO.PADBITS =2000
PAD.DEL =2000
PIA.PORT =E841
SKP.FF =2028
UART.IN =20DF
UD.TBL1 =2066

BIT.TIME =2001
DONE.UI =2103
EX.DEL =2000
GOTBAUD =2035
LP.FI =202D
LP.UI2 =20E9
LP1 =200E
MSK.IN =0040
NOT.THIS =2076
PAD.DELX =20BE
SET.BAUD =2002
SKP.UI1 =20F8
UART.OUT =2097
UD.TBL2 =208C

DEL0.5 =204D
END0.5 =205B
GET.BIT =2047
LOOPDEL1 =2061
LP.PDEL =20CA
LP.UI3 =20F1
LP2 =2013
MSK.OUT =0080
PAD.DEL =20C4
PIA.DIR =E843
SKP.BOUT =20DB
TBLBAUD =203F
UART.OUT1 =209E

;0000, 2109, 2109;
}
620 S. Peace Haven Road
Winston-Salem, N. C. 27103
WE WILL ASSIST WITH SPECIAL DESIGNS
(919) 765-2665  JOHNNY & HAZEL WEISNER
## ERROR CODES

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