

mae

For Commodore Computers - 8032, Super Pet,
2001 - 32K, 4001 - 32K, "64", etc.

- Text Editor
- Macro Assembler
- Enhanced Machine Language Monitor
- Relocating Loader
- Word Processor
- Scrolling Package
- IEEE Driver Library
- Macro Libraries

Serial Number 1729

This copy for IBM64

Disk Drive 1541

**Eastern
House**

MAE Macro Assembler/Text Editor for Commodore Computers

CONTENTS	PAGE
1. Introduction	4
2. Files Contained on the Diskette	7
3. Enhanced DOS Support Program	8
4. Micromon - An Enhanced Machine Language Monitor	9
5. Text Editor (TED) Features	17
A. Commands	17
B. Entry/Deletion/Change of Text	24
6. Assembler (ASSM) Features	25
A. Source Statement Syntax	25
B. Label File (or Symbol Table)	35
C. Assembling	35
D. Creating a Relocatable Object File	36
E. MACROS	38
F. Conditional Assembly	41
G. Interactive Assembly	44
H. Default Parameters on entry to ASSM	45
7. Relocating the Relocating Loader	45
8. Error Codes	45
9. String Search and Replace Commands	47
A. EDIT Command	47
B. FIND Command	49
10. Examples	49
A. TED	49
B. ASSM	51
11. Getting Started with MAE	53
12. MAE Scrolling Program	54
13. MAE Tape Interface	55
14. The MAE Simplified Text Processor (STP)	56
15. Program Development Aids	62
16. Memory Map	63
17. Special Notes	63

MAE Macro Assembler/Text Editor for Commodore Computers

18. Control Codes (for Serial Device)	65
19. Connection of Serial Device	66
20. ASSM/TED Users Group	66
21. Example Listing	67
A. Software Uart	67
B. Shell-Metzner Sort	67
22. Other Items for MAE owners	68
23. References	68

***** COPYRIGHT NOTES *****

A legal copyright protects the MAE Macro Assembler/Text Editor software and manual. No one may copy, reproduce, store in a retrieval system, or in any other way duplicate either the MAE software or this manual without written permission from EASTERN HOUSE SOFTWARE.

The buyer may make a backup copy of the MAE diskette for his own use. It is a Federal crime to copy the manual for the use of anyone other than the buyer or the person for whom a company bought this software. Buying this software conveys no license to manufacture, modify, or copy it in any manner. A violation of Federal Copyright Laws is any of the following:

- A) Copying the manual.
- B) Letting others copy either the software or the manual.
- C) Letting others use the diskette while retaining a copy.
- D) Using simultaneously more copies than the number bought.

A reward will be paid to anyone supplying information leading to the prosecution of persons violating this copyright.

EASTERN HOUSE SOFTWARE does not presume the buyer will violate copyright laws. Most do not. Some do, though, without realizing the consequences. Penalties and fines can be very severe for both individuals and companies who infringe upon this copyright. Software houses like ours incur tremendous expenses we cannot recover fully if illegal copying continues. The insufficient return on investment could also force the end of program maintenance and improvements.

EASTERN HOUSE SOFTWARE appreciates your buying our product. We hope you find it a valuable tool and a wise investment. If you should have any problems, please contact us at:

EASTERN HOUSE SOFTWARE
3239 Linda Drive
Winston-Salem, N.C. 27106 U.S.A.

1. INTRODUCTION

This manual describes the MAE (Macro Assembler/Text Editor) Software Development System for Commodore computers and shows you how to operate the system. The manual contains the MAE program structure, complete definitions of all commands, references, tables, and numerous examples. EASTERN HOUSE SOFTWARE (EHS) hopes you will be pleased with our MAE Software Development System and will find it a valuable programming tool.

This manual assumes you know assembly language programming fundamentals. The References section of this manual lists some good books which provide the information you need to learn assembly language. Also, you should have handy a manual that defines in detail the entire 6502 instruction set.

The MAE System evolved from a single program to a collection of powerful software development aids. This is why EHS calls MAE a Software Development System. MAE was designed to run on the following Commodore equipment:

CPU	Screen	Memory	Disk Drive
8032	80 Column	32K	4040, 8050, or PEDISK II
2001	40 Column	32K	4040, 8050, or PEDISK II
4001	40 Column	32K	4040, 8050, or PEDISK II
4001	80 Column	32K	4040, 8050, or PEDISK II
Super PET	80 Column	32K	4040, 8050, or PEDISK II
64	40 Column	64K	1541, 4040, or 8050

- Notes: 1-) If your CPU is a 2001 that was upgraded to 4.0 Basic, you should obtain the 4001 version.
- 2-) If your CPU is a 4001 that was modified to expand the screens display to 80 columns, you should obtain the 8032 version.
- 3-) The version of STCP that will work with the CGRS PEDISK II is available on 40 or 80 track 5 1/4 inch and IBM 8 inch diskette.
- 4-) MAE for the Commodore 64 (C64) will work with the 1541 disk drive or 4040/8050 drives if the CLINK 64 IEEE adaptor is used.
- 5-) Thus a version of MAE is available for practically any Commodore computer excepting the old "small keyboard" PET that has the original 1.0 ROMs, and the VIC 20.

MAE software will drive multiple disk drives and both Commodore printers like the 4022 and 1525E, and ASCII type printers like the C. ITOH

MAE Macro Assembler/Text Editor for Commodore Computers

Starwriter and Prowriter.

MAE, Macro Assembler/Text Editor, has evolved from a single program to a collection of powerful software development aids. This is the reason we refer to MAE as a Software Development System. The MAE Software Development System consists of the following programs: Coresident Macro Assembler/Text Editor, Enhanced DOS Support program, Word Processor, Screen Scrolling Package, IEEE Driver Subroutines, an extension to the Machine Language Monitor called Micromon, a Relocating Loader, and numerous other utilities. To properly use this system, you load in the proper software to perform the intended task. For instance, to do assembly language program development one would load the MAE Assembler/Text Editor and Micromon. To prepare program documentation, one would load the MAE Assembler/Text Editor and the Word Processor. MAE can also be personalized. If you like the capability to scroll thru MAE files using the cursor up/down key, then you should load in the Scrolling Program.

MAE includes an enhanced DOS Support Program to aid you in loading these programs using special commands. For example, the command LA will load the Assembler/Text Editor, LM will load Micromon, etc. We have tried to make MAE as flexible and powerful as can be. We are trying to make MAE the "Compatible Assembler" by making the software upward compatible with our less expensive cassette assembler. In fact, we have included a program so MAE users can exchange source software with the PET cassette-based users. MAE is also available for the Apple and ATARI computers. The PET, Apple, and ATARI MAE along with our cassette-based assemblers for PET/Apple/ATARI/KIM/SYM/etc. have similar source syntaxes to help standardize assembly programs so they can be easily transferred from one 6502-based computer to another type.

We hope you will be pleased with the MAE Software Development System and find it of immense value in your program development activities.

To fully understand how to use MAE, we suggest that you read the first few parts of this manual until you get tired or anxious to get started. Then jump down to the part titled getting started with MAE. Next, try the part titled "Examples". Then, you should have the general idea how MAE works and should finish reading the entire manual. Of course, if you are experienced at using an assembler/editor and in a hurry, you can jump right in to using MAE and just refer to the manual when you have a problem. Learning is a personal experience and only you can know how you learn the fastest. So, you do it the way you think best!

One of the first things you should do is read the MAE.NOT file. This is a file of important notes and information discovered after this manual was printed.

```
To read MAE.NOT, type:  ]GET "MAE.NOT"
                        ]PRINT
```

Use the RUN/STOP and RETURN keys to stop and continue the listing.

MAE Macro Assembler/Text Editor for Commodore Computers

The Macro Assembler (ASM) and Text Editor (TED) resides simultaneously in 10K bytes of memory (5000-77FF). The collective assembler and text editor is referred to as the MAE Assembler/Text Editor.

As mentioned, the MAE ASM/TED 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 = \$2000-\$2FFC
- . Relocatable Object Buffer = \$7800

These boundaries leave memory for the extended monitor (\$1000-\$1FFF), the DOS support at upper memory (\$7C00-\$7FFF), and memory for Basic and Machine Language programs (\$0400-\$1000, C64: \$0800-\$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 ASM/TED 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 Disk Drive.

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

Initial entry (or cold start) to MAE is at address \$5000. If the

MAE Macro Assembler/Text Editor for Commodore Computers

break command (JBR) 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 JBR, 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, C64: \$37, \$38) to point to just below MAE and its text and label files. For example, to protect memory above \$2000, change HIMEM to: 00 20.

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. 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:

Filename	Description
MAE/DOS.EXE	Enhanced DOS Support Program (Wedge)
MICROMON.EXE	Enhancement to PET M.L. Monitor
MAE.EXE	MAE object code
REL.EXE	Relocating Loader object code
REL.REL	Relocating Loader relocatable code
PET.LIB	* Library of PET ROM locations
MAE.NOT	* Some notes on MAE
WORDP.EXE	Word Processor Program

MAE Macro Assembler/Text Editor for Commodore Computers

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 (Use with PET16, see #20 and #25 issues of Micro Magazine)
IEEE.LIB	* IEEE Machine Language Driver Routines
SECTOR.CTL	* Example program which illustrates use of IEEE.LIB - displays disk sector
SECTOR.PGM	
UART.CTL	* Example Program (Software Uart Driver)
UART.MO1	* "
UART.MO2	* "
UART.MO3	* "
GL.SORT.CTL	* Shell-Metzner Sort Example
GL.SORT.MO1	* "
SM-SORT.ASM	* "
SORT.TEST.BAS	Basic test program for the sort

* = 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
>>n	To change device number - i.e. >>9 allows access to device 9.

In addition to these commands, the MAE/DOS.EXE program provides an auto-repeat key feature and a number of support commands.

The support commands are a number of two letter commands of the form xy that may be entered to quickly and conveniently load and execute various programs on the MAE disk.

x specifies a function:	L = Load a program
	C = Cold start a program
	W = Warm start a program
	E = Load and then cold start a program

MAE Macro Assembler/Text Editor for Commodore Computers

(This is the Execute Function.)

y specifies the program: A = MAE Assembler/Text Editor

M = Micromon Monitor

W = Word Processor

S = Scrolling Program

T = Tape Program

L = Loader Program

U = User Program

(Filename must begin with "USER
must load in memory at \$0500,
(C64: \$0900), cold start at that
address, and warm start at
\$0503 (C64: \$0903).

For example, to load the MAE ASM/TED, Micromon, and the Scrolling Program, we would enter: LM - load Micromon

LS - load Scroll Program

EA - load and run the MAE ASM/TED

Of course, we could also have entered: LM, LS, LA, CA.

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:

EA

The difference is 20 keystrokes versus just 3!

4. MICROMON - An Enhanced Machine Language Monitor

The file on the diskette named MICROMON.EXE is an enhanced machine language monitor originally developed by Bill Seiler and later enhanced by Arthur Cochrane. Micromon traces its origin to a program called EXTRAMON also developed by Mr. Seiler. When we heard of these machine language monitors, we contacted the authors and asked if we could include them with the MAE. They graciously agreed. We believe that MAE users will find that these monitors are of great help in debugging machine language programs and for just exploring the the internal workings of the Commodore computers.

MAE Macro Assembler/Text Editor for Commodore Computers

Micromon occupies 4K of memory at \$1000-\$1FFF. Cold start entry is at \$1000. On the C64, Micromon must be cold started before the MAE]BREAK command will function. If you press RUN/STOP and RESTORE on the C64, this will disable Micromon and require you to cold start at \$1000 again.

If by chance you are interested in obtaining the source listing to Micromon, please contact the ATUG Users Group mentioned later in this manual.

Micromon contains 27 commands which aid in program development and incorporates the ability to use the cursor up and down cursor control keys to scroll the memory dump and disassembler outputs.

The commands are described below:

SIMPLE ASSEMBLER

```
.A 2000 A9 12    LDA #$12
.A 2002 9D 00 80 STA $8000,X
.A 2005 DEX:GARBAGE
```

In the above example, the user started assembly at 2000 HEX. The first instruction was load a register with immediate 12 HEX. In the second line, the user did not need to type the A and address. The simple assembler retyped the last entered line and prompts with the next address. To exit the assembler, type a return after the address prompt. Syntax is the same as the Disassembler output command. A colon (':') can be used to terminate a line and insert a comment.

BREAK SET (Not available on C64)

```
.B 1000 00FF
```

The example sets a break at 1000 HEX on the FF-th occurrence of the instruction at 1000. Break set is used with the Quick Trace command. A Break Set with no count entered, stops on the first occurrence.

COMPARE MEMORY

```
.C 1000 2000 3000
```

Compares memory from HEX 1000 to HEX 2000 to memory beginning at HEX 3000. Locations with unequal bytes will be printed on the screen.

DISASSEMBLER

MAE Macro Assembler/Text Editor for Commodore Computers

.D 2000 3000

```
., 2000 A9 12    LDA #$12
., 2002 9D 00 80 STA $8000,X
., 2005 AA       TAX
```

Disassembles from address 2000 to 3000. The three bytes following the address may be modified. To modify a byte, simply cursor to the byte to be changed, type the new byte, and press return. The disassembly mnemonic will be changed to reflect the new byte.

Disassembly can be performed under control of the cursor up and down keys. To disassemble one byte at a time, type:

.D 1000

If the cursor is on the last line, one instruction can be disassembled for each press of the cursor down key. If the cursor down key is held down, disassembly will be continuously outputted. Disassembly can even be in reverse. If the screen is full of a disassembly listing, place the cursor at the top line of the screen and press the cursor up key.

EXIT MICROMON

.E

This command kills Micromon and exits to Basic.

FILL MEMORY

.F 1000 1100 FF

Fills memory from 1000 HEX to 1100 HEX with the byte FF.

GO RUN

.G

Go to the address contained in the PC Register. All registers will be replaced with the displayed values.

.G 1000

Go to address 1000 HEX.

MAE Macro Assembler/Text Editor for Commodore Computers

HUNT MEMORY

.H C000 D000 ^READ

Hunt thru memory from C000 HEX to D000 HEX for the ASCII string READ and print the address where it occurred. A maximum of 32 characters may be used.

.H C000 D000 20 D2 FF

Hunt memory from C000 HEX to D000 HEX for the sequence of bytes 20 D2 FF and print the address where they occurred. A maximum of 32 bytes may be used. A hunt may be aborted by pressing the STOP key.

KILL MICROMON

.K

Restore the Break and IRQ vectors to values before Micromon was initialized, and break to the standard PET machine language monitor. Micromon can be reinitialized via a GO to the location contained in the PC register.

LOAD

.L "RAM TEST",08

Load the program named RAM TEST from disk.

MEMORY DISPLAY

.M 0000 0008

.: 0000 30 31 32 33 34 35 36 37 01234567
.: 0008 38 39 41 42 43 44 45 47 89ABCDEF

Display memory from 0000 HEX to 0008 HEX and also the ASCII values of each location. The bytes following the address may be modified by editing and then pressing return.

Memory display can also be controlled using the cursor up and down keys.

NEW LOCATER

MAE Macro Assembler/Text Editor for Commodore Computers

```
.N 7000 77FF 6000 0400 9000
.N 77CD 77FF 6000 0400 9000 W
```

Relocate a machine language program to a new address. The first line fixes all three byte instructions in the range of 7000 HEX to 77FF HEX by adding 6000 HEX offset to the bytes following the instruction. New locator will not adjust any instruction outside of the 0400 to 9000 HEX range. The second line adjusts .WORD values in the same range as the first line. New locator will stop and disassemble any bad opcodes.

CALCULATE BRANCH OFFSET

```
.O 033A 033A FE
```

Calculate the offset for branch instructions. The first address is the starting address and the second address is the target address. The offset is then displayed.

QUICK TRACE (Not available on the C64)

```
.Q
.Q 1000
```

The first example begins a machine language trace at the address in the PC register. The second begins the trace at 1000 HEX. Each instruction is executed as in the WALK command but no disassembly is displayed. The Break Address is checked for the break on the Nth occurrence. The trace may be aborted by pressing the STOP and '=' (left arrow on business keyboard) keys at the same time.

REGISTER DISPLAY

```
. R
  PC  IRQ  SR  AC  XR  YR  SP
.; 0000 E455 01 02 03 04 05
```

Displays the register contents saved when MICROMON was entered. The contents may be changed by cursoring up, overwriting with a new value, and then pressing RETURN.

SAVE

```
.S "1:PROGRAM NAME",08,0800,0C80
```

MAE Macro Assembler/Text Editor for Commodore Computers

Save to disk drive #1 memory from 0800 HEX up to but not including 0C80 HEX under filename PROGRAM NAME.

Note: On the C64, the format is .S "1:PROGRAM NAME",0800,0C80,08
The device number is on the end and may be left off for a default of 08.

TRANSFER MEMORY

.T 1000 1100 5000

Transfer memory in the range 1000 HEX to 1100 HEX to a new location starting at 5000 HEX.

WALK CODE (Not available on C64)

.W

Single step starting at the address in the PC register.

.W 1000

Single step starting at address 1000 HEX. Walk will cause a single step to execute and will disassemble the next instruction. Press the STOP key to stop the walking. The 'J' key is useful at a JSR instruction. It causes the subroutine to be completely executed before stopping at the instruction immediately after the JSR.

EXIT to BASIC

.X

Return to BASIC READY mode. The stack value saved when entered will be restored. Care should be taken that this value is the same as when the MONITOR was entered. A CLR Basic command will fix any stack problems.

CHANGE CHARACTER SETS

.Z

Change from uppercase/graphics to lower/uppercase mode or vice versa.

HEX CONVERSION

.\$4142 16706 a b 0100 0001 0100 0010

MAE Macro Assembler/Text Editor for Commodore Computers

A HEX number is input and the decimal value, the ASCII representation for the two bytes, and the binary values are returned.

DECIMAL CONVERSION

.#16706 4142 a b 0100 0001 0100 0010

A decimal number is input and the HEX value, the ASCII representation for the two bytes, and the binary values are returned.

BINARY CONVERSION

.%0100000101000010 41442 16706 a b

A binary number is input and the HEX value, the decimal value, and the ASCII representation are returned.

ASCII CONVERSION

."A 41 65 0100 0001

An ASCII character is input and the HEX value, decimal value, and binary values are returned.

ADDITION

.+ 1111 2222 3333

The two HEX numbers inputted are added and the sum is displayed.

SUBTRACTION

.- 3333 1111 2222

The two HEX numbers inputted are subtracted and the difference is displayed.

CHECKSUM

.& A000 AFFF 67E2

The checksum between the two inputted addresses is calculated and displayed.

MICROMON INSTRUCTION SUMMARY

A Simple Assembler
B Break Set
C Compare Memory
D Disassembler
E Exit Micromon
F Fill Memory
G Go Run
H Hunt Memory
K Kill Micromon
L Load
M Memory Display
N New Locator
O Calculate Branch
Q Quick Trace
R Register Display
S Save
T Transfer Memory
W Walk Code
X Exit to Basic
Z Change Character Sets
\$ Hex Conversion
Decimal Conversion
% Binary Conversion
" ASCII Conversion
+ Addition
- Subtraction
& Checksum

5. 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 (PET versions only).

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 "]" 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]PR may be several non-space characters in length, MAE only considers the first two. For example,]PR,]PRI,]PRINT, and]PRETTY will be interpreted as the print command.

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

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 "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:

]ALPHA

Toggle shift character set from graphics to lower case and vice versa by toggling PET control at \$E84C (C64: \$D018).

]ASSEMBLE file w

If file is specified, load the file into text file and then begin assembly with contents of text file.
If file is not specified, then assemble contents of text file.

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

]AUTO x

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

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

Example: Output directory is]DC "\$"
Scratch file TEST is]DC "S:TEST"

Note: Entry of]DC 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 9.

]FIND t S1 t

String search.
See part 9.

]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 "1:CRTDVR" 1000
]GET "0: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 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. First pass must have
been previously performed. If file is entered then the
text file is loaded before executing the second pass,
else]PASS will assume the file is in the text file.

]PRINT x y

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

]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.

]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. Zero page is preserved on exit from MAE
and restored on reentry.

]SET ts te ls le bs

If no parameters are given, the text file, label file,
and relocatable buffer boundaries (addresses

indicating text file start, end, label file start, end, and relocatable buffer start) will be output on the first line. On the second line the output consists of the present end of data in the text and label file. This command is commonly used to determine how much memory is remaining in the text file. If you are inputting hex digits for these addresses, 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).

]TI 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]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 PET.

]TO 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:

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 `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 `]CL` command.

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

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

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

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

Text may be entered more easily by use of the auto line numbering feature (`]AU` command). Any `]AU x` where x does not equal 0 puts the TED in the auto line number mode on the next entry of a line number. 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 JFO 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.

6. 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 JOU), may be performed in order to generate a relocatable object file of the program in the text file. This file is recorded on disk and may be relocated at the users descretion practically anywhere in memory.

A. Source Statement Syntax

Each source statement consists of 5 fields as described below:

line number	label	mnemonic	operand	comment
-----	-----	-----	-----	-----

Label:

The first character of a label may be formed from the following characters:

` A thru Z [\] _

While the remaining characters which form the label may be constructed from the above characters and the following characters:

. / 0 thru 9 : ; < > ?

The label is always entered immediately after the line number.

Mnemonic (or Pseudo Op):

The mnemonic or pseudo op is separated from the label

by one or more spaces and consists of a standard 6502 mnemonic of table A, pseudo op of table B, or macro name.

Operand:

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

Comment:

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

NOTE: It is 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)

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

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` The formatter (i.e. JFORMAT SET) alligns fields in neat columns. Unfortunately, if you enter a number of leading spaces in the `.BY` pseudo op such as `.BY ' TEST'`, it will be formatted as `.BY 'TEST'`. To get arround this, issue a null string any time you want leading spaces such as `.BY '' 'TEST'`.

.CE

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

.CT

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

Note: Only one `.EN` pseudo op is allowed in each assembly and if `.CT` is used, the `.EN` must be at the end of that file. Thus, files referenced

via .FI must not have a .EN pseudo op.

label .DE label exp.

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

label .DI label exp.

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

.DS label exp.

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

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

.EC

Suppress output of macro generated object code on source listing. This is the default state. See part 6E.

.EJ

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

.EN

Indicates the end of the source program.

.ES

Output macro generated object code on source listing.
See part 6E.

.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 ? followed with space and then accept 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 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 6E.

.ME

Macro end of definition. See part 6E.

.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 #X1101	;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
 ;(May be omitted for ASSM to determine if
 ;zero-page or not.)

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 JOU 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 pass for a complete assembly. When the end of a pass is encountered, MAE will output the message END MAE PASS!. If for some reason you terminate the assembly on the second pass, you may restart at the beginning of the second pass using the JPASS command.

D. Creating a relocatable object file (JOU)

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

Also, the .SE stores a two byte external address and the .SI stores a two byte internal address. 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 (i.e. JSR \$FFD2 to output to the screen) 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 PET operating system uses. Examples are the current cursor position, the IEEE first and second address storage locations, etc.

An example using an internal address would be when you want to store the address of a number of subroutines contained in the program in a table. To illustrate, MAE's command mnemonics and associated processing routines address's are stored in a table as follows:

```
.BY ^CO^          ;copy command
.SI COPY_MOD      ;copy module address
```

MAE Macro Assembler/Text Editor for Commodore Computers

```
.BY 'MO'           ;move command
.SI MOVE MOD       ;move module address
.BY 'PR'           ;print command
.SI PRINT MOD      ;print module address
```

Note that if MAE were relocated, the COPY, MOVE, and PRINT modules would move also. That is why these are internal addresses so the relocating loader will adjust them when the program is relocated. 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. Also suppose that you referenced BUFFER via LDA BUFFER+3. BUFFER is an internal address and '3' is nonsymbolic or external. When combined, the expression becomes internal. Thus the relocating loader will adjust BUFFER+3 when the program is relocated. Conversely, 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. Note: The program will not be relocated properly until it has been assembled at \$2000 (via .BA \$2000). Next assemble the program via JAS, and then issue the JOUT 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 (C64: \$8500) if in the monitor or SYS 1280 (C64: SYS 34048) 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 references.
- 4) PGM EXE OFFSET? Address the program is to execute.
- 5) PGM STORE OFFSET? Address to store the program object code.

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

As an example, lets 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? D8 "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
.
.

```

MAE Macro Assembler/Text Editor for Commodore Computers

```
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 or appear in a Macro Global file (see .MG Pseudo OP). Each file is assigned a unique file sequence number (2**16 maximum files in each assembly) which is assigned to each macro name. Thus the same macro can appear in more than one file without causing a !06 error. If a macro with the same name is defined twice in the same file, then the !06 error will occur.
- 3) Macros may be nested up to 32 levels. This is a limitation because there is only so much memory left for use in the stack.
- 4) If a macro has more than one parameter, the parameters should be separated using spaces - do not use commas.
- 5) The number of dummy parameters in the macro definition

must match exactly the number of parameters in the call for expansion.

- 6) The dummy parameters in the macro definition must be symbolic labels. The parameters in the expansion may be symbolic or non-symbolic label expressions.
- 7) If the .ES pseudo op is entered, object code generated by the macro expansion will be output in the source listing. Also, comment lines within the macro definition will be output as blank lines during expansion. Conversely, if .EC was entered, only the line which contained the macro call will be output in the source listing.
- 8) A macro name may not be the same as a 6502 mnemonic, pseudo op, or conditional assembly operator.

F. Conditional Assembly

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

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

IFE label exp.

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

IFN label exp.

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

IFP label exp.

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

IFM label exp.

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

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

SET label=label exp.

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

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

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

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

              IFE  CHAR.LINE-64
```

```
;CODE FOLLOWS FOR 64 CHARACTER PER LINE
```

```
.
.
***
```

```
IFB 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, and only partly expanded for 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
IFB EXPAND
JSR SORT.CALL ;CALL SORT
***
```

```
IFB 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 have also been used in our CRT Controller software example as follows:

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

Note in the above that the label CHAR.LINE was defined on the same line as the .IN. This is perfectly OK and alleviates the need to define CHAR.LINE on a separate line using the .DE. You might say, "It kills two birds with one stone!!"

There are many other 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)

7. RELOCATING THE RELOCATING LOADER

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

To relocate the relocating loader, load the executable copy (REL.EXE) and begin execution. When FILE NAME? is output, enter "REL.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 relocater 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 (C64: \$8500) as for the executable copy supplied (REL.EXE).

8. 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 or file not found.
16
15      Syntax error in JED command.
14      Device numbers 0,1,2,3 not allowed.
13      Multiple .CT assignment.
12      Command syntax error or out of range error.
11      Missing parameter in JNU command.
10      Overflow in line # renumbering.
        CAUTION: You should properly renumber the
        the text file for proper command operation.
OF      Overflow in text file - line not inserted.
OE      Overflow in label file - label not inserted.
OD      MAE expected hex characters, found none.
OC      Illegal character in label.
OB      Unimplemented addressing mode.
OA      Error in or no operand.
O9      Found illegal character in decimal string.
O8      Undefined label (may be illegal label).
O7      .EN pseudo op missing.
O6      Duplicate label.
O5      Label missing in .DE or .DI pseudo op.
O4      .BA or .MC operand undefined.
O3      Illegal pseudo op.
O2      Illegal mnemonic or undefined macro.
O1      Branch out of range.
OO      Not a zero page address.
ED      Error in command input.

```

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

ERROR CODE	DESCRIPTION
2F	Overflow in file sequence count (2**16 max.)
2E	Overflow in number of macros (2**16 max.)
2D	
2C	
2B	.ME without associated .MD
2A	Non-symbolic label in SET pseudo op.
29	Illegal nested definition.
28	
27	Macro definition overlaps file boundary.
26	Duplicate macro definition.
25	Quantity parms mismatch or illegal characters.
24	Too many nested macros (32 max.)

```

23      Macro definition not complete at .EN
22      Conditional suppress set at .EN
21      Macro in expand state at .EN
20      Attempted expansion before definition.

```

9. 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 tS1tS2t %d * x y
          #

```

Where: t is a non-numeric, non-space terminator
 S1 is the string to search for
 S2 is the string to replace S1
 d is don't care character. Precede 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 Edit Form 2 (see below)

```

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:

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

To simply delete all occurrences of LOOP, enter:

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

If you want to delete all references of LOOP01, LOOP02, LOOP03, etc., enter:

```
]EDIT /LOOP%Z%/ 100 600
```

Note that the `%` character is a wild card or don't care character. To do the same thing as above but not display changes on the screen, enter:

```
]EDIT /LOOP%Z%/ # 100 600
```

Suppose for example that there are some occurrences of the label BUFFER that you want to change to CRT/BUFFER, some you want to just delete the whole line, and other times you want to extensively edit the line. This requires a conditional replacement capability with you as the decision maker.

To do this, enter:

```
]EDIT &BUFFER&CRT/BUFFER& *
```

(Note that the string terminator is & because \ is a string character. The string terminator is always defined as the first non-space character after the EDIT mnemonic.)

Now, each time MAE's edit command encounters the string BUFFER, the line will be printed on the screen and an asterisk (*) prompt will appear awaiting your entry of one of the subcommands (A, D, M, S, X, 2) described above. For example, to replace BUFFER with CRT/BUFFER, press "A". To delete the whole line, press "D". To extensively edit the line, press "2". 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.

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

At the end of the]EDIT operation, the number of 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 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]FIND command. Its form is:

]FIND tSl# x y

Where: t, Sl, #, x, and y are as defined in 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 %/#

10. EXAMPLES

A. TED Examples

- #1 Illustrate 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 (MICROMON).

/MICROMON.EXE or MC
SYS 4096

- #3 Illustrate entry of text.

```

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

```

#4 Illustrate listing of text.

```

]PRINT
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.
]PUT D9 "O:TEST"

#6 Get file from disk (device 8, drive 1) named TEST.
]GET "O:TEST"

Note: The default is device 8.

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

#8 Output directory for drive 0.
]DC "\$0"

#9 Scratch file TEST.
]DC "S:TEST"

#10 Read disk error channel.
]DC

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

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

#13 Assign serial device 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 return via SYS 20480 (cold start)
 or SYS 20483 (warm start)
- #23 Go to Machine Language Monitor.
]BREAK return via G 5000 (cold start)
 or G 5003 (warm start)
- #24 Run assembly program at symbolic label BOX.
]RUN BOX
- #25 Change from Uppercase/Graphics character set to
Lowercase/Uppercase or vice versa.
]ALPHA

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.

MAE Macro Assembler/Text Editor for Commodore Computers

```
.BA $1000      - Note these are order dependent and
.MC $4000      - must be entered in this order.
.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 !OF error on subsequent
loads, you should know that you have overflowed
the text buffer. The solution is to allocate
more memory (via]SET command) and then
reassemble.

#11 Show example of a very long label.
MEMORY.TEST.FOR.6502
JMP MEMORY.TEST.FOR.6502

NOTE: Long labels (greater than that specified via
]FO 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
```

```
      .
      .
      JSR  RDT
```

-- OR --

```
      JSR  $FFCF
```

11. GETTING STARTED WITH MAE

Load MAE/DOS, MAE, and MICROMON 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 additional DOS commands.
 These commands are aids to quickly load and transfer control to other MAE programs.
- 4-) Type EM to load and initialize the extended monitor.
- 5-) Type .X to return to BASIC.
- 6-) Type EA to load and cold start the MAE Assembler/Text Editor.

MAE will respond with:

```
C 1979 BY C MOSER
```

```
3000-4FFC  2000-2FFC  7800
3000  2000
```

```
]
```

This displays the default allocations of memory for the text file (3000-4FFC), label file (2000-2FFC), 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 \$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 MICROMON occupies memory at \$1000-\$1FFF and MAE occupies \$5000-\$77FF. MICROMON also sets the Basic variable HIMEM to \$1000 to indicate the end of memory available for Basics use. This in effect protects MICROMON, 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:

C64 Address	PET ADDRESS	DATA
-----	-----	----
\$0037	\$0034	\$00
\$0038	\$0035	\$80

The first thing you should do now is to load the MAE.NOT file via
]GET "MAE.NOT"
]FORMAT CLEAR - turn formatting off
]AL - 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 11.

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

12. MAE Scrolling Program

SCROLL is a program enhancement which provides for forward and reverse scrolling thru MAE source files using the cursor up (CU) and cursor down (CD) keys.

LOAD and INITIALIZE

- 1- Go to Basic level
- 2- Type: LS
 EA or WA or CA whichever is appropriate

]RUN \$7800 (\$8000 on the C64)

OPERATION

If the cursor is in columns 0 or 1: CU at top of screen results in a reverse scroll thru the file until text file start. CD at bottom of screen results in scroll forward until text file end.

SCROLL Notes

- 1- The .EXE file occupies memory in range \$7800-\$7C00 (\$8000-\$8400 on C64).
- 2- If you use the Relocating Loader at the same time you are using SCROLL, insure that the buffer is not set for \$7800 thru \$7C00 (C64: \$8000-\$8400).
- 3- Scrolling is temporarily disabled on all exits from MAE, but reenabled on warm start.
- 4- Scrolling is permanently disabled on cold start but may be reenabled via]RUN \$7800 (C64: \$8000).
- 5- The SCROLL program may be relocated using the Relocating Loader and Scrolls .REL file. Enter 0 for Z-PAGE and ABS-PAGE offset. The initialize address will change if relocated.
- 6- Scrolling will not function unless the cursor is in the two left-most columns (columns 0 and 1).

13. MAE Tape Interface

This program provides MAE the capability to read and write ASM/TED source files to cassette tape. ASM/TED is EHS's cassette based assembler. The purpose of the Tape Interface program is to provide the capability to transfer MAE source files to cassette based users.

LOAD and INITIALIZE

- 1- Go to Basic level
- 2- Type: /O:TAPE???.exe
EA or WA or CA whichever is appropriate

To Read from Tape

Type]RUN \$7900, insert cassette, and press PLAY.

To Write to Tape

Type]RUN \$7903, insert cassette, and press PLAY and RECORD.

MAE Tape Interface Notes

- 1- The .EXE file occupies memory in range \$7900 thru \$7AFF.
- 2- The MAE tape interface program may be relocated using the Relocating Loader and associated .REL file. Enter 0 for Z-PAGE and ABS-PAGE offset. The read/write addresses will change if relocated.

14. 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 easily remembered word processing functions, and usage of an already familiar text editor to enter and edit the raw text.

STP, unlike some other word processor 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, insert STP 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, words are not combined to fill to the specified margins. This is useful for tabular material, tables, and other text that you want to appear exactly as you typed it in.

BEGIN A NEW PAGE (.ff)

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

LITERAL SPACE (character)

Normally, spaces are not processed like other characters. If several spaces are entered consecutively, the STP word processor 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. 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. using the .nofill macro.

RAGGED RIGHT MARGIN (.rr)

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

RAGGED LEFT MARGIN (.rl)

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

SKIP NEXT N LINES (.ln)

Use this macro to skip a number of lines before printing the next line of text. For example, to skip 2 lines and begin printing, enter .l2. If you enter .1 by itself, one will be assumed. Thus .1 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.

MAE Macro Assembler/Text Editor for Commodore Computers

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 of this macro is for paragraph numbering. An example for this is ".pi 1.01". In this example text would appear similar to the following:

```
1.01 .....  
      .....  
.....  
.....
```

Note that the second line indents so that the number 1.01 stands out. The number of lines to be indented can be controlled by the "d" parameter in the .p macro above.

PAGE TITLE (.t# title text)

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

PAGE FOOTERS (.foot# foot text)

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

FILE LINKAGE (.link "filename")

This macro can link text modules together as a single printable document. An example is .link "SECTION-14". Simply enter the link macro at the bottom of the file immediately before the one you want loaded and processed next. 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. Note that the maximum document length that this word processor can handle is dependent on the amount of disk space - thus there is capacity for tremendously large documents.

RUNNING MACHINE LANGUAGE PROGRAMS (.ru \$xxxx)

This macro allows the user the capability to execute a machine language program at any point during the text formatting process. This is useful when it is desired to send command strings to intelligent printers like the Prowriter printer to change type fonts, to perform elongated printing, etc. The machine language program should perform its function and then execute an RTS instruction to return control back to the word processor. The text processing then continues with no indication of the interruption.

PRINTING TEXT -]WC and]WP

Normally]RUN \$500 (C64: \$8500) is used to send STP output to the CRT, and]RUN \$503 (C64: \$8503) is used to send output to the printer. The STP also contains an initialization routine at \$506 (C64: \$8506) which provides two commands. After the initialization routine is run by typing]RUN \$506 (C64: \$8506), 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]WC and]WP commands are entered, the .link macros are ignored unless specifically instructed via:]WP LINK. Additionally, the first n pages may be suppressed by typing]WP LINK n or]WP n. These options can also be used with the]WC command.

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

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

The .shape 2 macro may be used to create a user defined shape. In order to define the desired shape, .set macros are used to make entries in

the user shape table corresponding to the desired shape. The parameters in the .set n 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 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 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 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

MAE Macro Assembler/Text Editor for Commodore Computers

- 1) Load the word processor and MAE via:
 - LW - Load the word processor
 - LS - Load Scrolling program (Optional)
 - EA - Execute the MAE Assembler/Editor

Note: Basics Hi Mem pointer is set above the word processor. On the PET versions, either poke this pointer to \$500 or refrain from using Basic commands which use program memory space.

- 2) Initialize the word processor via]RUN \$506 (C64: \$8506). This adds two new commands:]WC and]WP.
- 3) Initialize the scrolling program (optional) via]RUN \$7800 (C64: \$8000).
- 4) Enter upper case/lower case mode via the]AL command. Clear format mode via]FORMAT CLEAR.
- 5) Enter raw text using MAE for editing. Include all necessary text processing macros.
- 6) When you are finished entering the raw text and associated text macros, generate a formatted output via:

WC < for output to screen only
WP < for output to screen and Printer

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 WC (CRT only) or WP (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 screen, it will not appear neat on the screen 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.

15. Program Development Aids

The MAE Software Development System consists of the following program development aids:

MAE Macro Assembler/Text Editor for Commodore Computers

- PET.LIB - MAE library file containing a list of PET ROM locations.
- IEEE.LIB - MAE library file containing a number of IEEE driver subroutines.
- MLMACROS.MLIB - MAE library file containing a number of macros for machine language program development.
- SWEET16.MLIB - MAE library file containing a set of SWEET16 macro instructions. SWEET16 is a pseudo 16-bit processor originally developed for the Apple II computer but later adapted for Commodore computers.
- SCROLL.REL - Relocatable copy of the Scroll program.
- TAPE.REL - Relocatable copy of the Tape program.
- REL.REL - Relocatable copy of the Relocating Loader.

16. Memory Map

The following is a memory map of the MAE Software Development System:

(C64)	(PET)	SIZE	USAGE
-----	-----	----	-----
1000-1FFF	1000-1FFF	4K	MICROMON
2000-2FFC	2000-2FFC	4K	Label Buffer
3000-4FFC	3000-4FFC	8K	Text Buffer
5000-77FF	5000-4FFC	10K	MAE
7C00-7FFF	7C00-7FFF	1K	Enhanced DOS Support
7800-78FF	7800-78FF	.25K	Relocating Buffer
8000-83FF	7800-7BFF	1K	Scroll Program
8500-8FFF	0500-0FFF	2.8K	Word Processor or Relocating Loader

17. SPECIAL NOTES

- * Always define all zero-page references BEFORE you reference them. A good practice is to make all definitions at the start of the source file. If you do not define the zero-page labels before you reference them, an assembly phase error between Pass 1 and Pass 2 will occur. This happens because the assembler will default to absolute (3-byte) addressing mode for all absolute and also for all undefined operands. If an undefined operand was later defined as zero-page, then during Pass 2 of assembly the instruction

will be assembled as 2-bytes causing a difference in references to labeled locations. There is no error message for these Phase Errors.

- * 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 (\$34, \$35, C64: \$37, \$38) so Basic will not clobber MAE, or its text or label files.
- * Use quality diskettes like Elephant Memory. 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

- * If you accidentally clear a text file (via]CLEAR), you can "unclear" the text file via the following procedure:
a) Type]PRINT / to display last line in text file.

- b) Type `JBREAK` to go to the machine language monitor.
 - c) Use the monitors `.M` command to copy the contents of locations \$7655, \$7656 to locations \$764B, \$764C.
 - d) Add 2 to the text file start address (determine via the `JBSET` command), and set that location to hex 20. For example, if the start of the text file is \$3000, set location \$3002 to hex 20.
 - e) Reenter MAE. The first line may be in error but the rest should be OK. The second line will also be in error if the first line contained just a single character.
- * Some of the support files on the MAE disk were set up to use drive #1 as in `.FI "1:name"`. If you have only one drive, it will be drive #0 and I/O to drive #1 will result in `"74,DRIVE NOT READY,00,00"`. If this occurs, simply change the "1" to "0". A quick way to do this is via the `EDIT` command: `JEDIT /"1:"0:/.`
- * Some of the support files require more text file space than is allocated on cold start. When the example program `SECTOR.CTL` is assembled, it will require more than the default text file space. For these cases, use the `JBSET` command to expand the text file space.

18. CONTROL CODES (for 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 `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
<code>^</code>	Null (hex 00)
<code>B</code>	Restore zero page and go to Basic
<code>C</code>	Restore zero page and go to Monitor
<code>G *</code>	Bell
<code>H *</code>	Backspace (delete previously entered char.)
<code>I *</code>	Horizontal tab to next 8-th char. position
<code>J *</code>	Line feed
<code>M *</code>	Carriage return

MAE Macro Assembler/Text Editor for Commodore Computers

O Continue processing but no output (same as DEL)
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.

19. 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]TI and]TO are provided to direct MAE to input or output on this serial port.

20. ASSM/TED USERS GROUP

An ASSM/TED Users Group has been formed 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 (about \$10.00 per disk) 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.
MICROMON - Extension to PET Machine Language Monitor.

For more details, contact:

ATUG Disk Exchange
c/o Brent Anderson
200 S. Century
Rantoul, Illinois 61866
(217) 893-4577

(SOUTHERN SOLUTIONS)

21. EXAMPLE LISTING

Two examples of programs in MAE's syntax follows. One 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 remuneration.

The second program is a Shell-Metzner sort program which will sort basic string arrays A1\$, A2\$, A3\$, and A4\$. The basic program passes a number of parameters one of which is which array to sort on. The Shell-Metzner sort program will sort the specified array and relatively exchange the elements of the other arrays. In this way, one could assign last names to the A1\$ array, first names to the A2\$ array, cities to the A3\$ array, zip codes to the A4\$ array, and sort on any of the four items. The specific details on the requirements of the basic program which calls the sort program are commented in the MAE file GL.SORT.M01. The basic program SORT.TEST.BAS illustrates how to interface to the sort program.

The Shell-Metzner sort program source code is contained on the supplied diskette under files: GL.SORT.CTL, GL.SORT.M01, and SM-SORT.ASM. This source is designed to use the four A\$ arrays. If you want to design your own special purpose sorting algorithm, you may want to design it around the GL.SORT or around the core Shell-Metzner sort algorithm contained in SM-SORT.ASM.

22. Other Items of use for MAE Owners

Eastern House has a number of items that may be of use to you in your development of programs. They are:

- a) EPROM Programmer - Hardware/Software that operates off the user port and cassette port. This programmer programs 2716 and 2532 EPROMs. Housed in an attractive cabinet. Easy to use. \$75.00.
- b) EPROMs and diskettes can be obtained from Eastern House at discount prices. The cost of these items change constantly - please call for current price.
- c) TRAP65 is a hardware device that plugs into the 6502 microprocessor chips socket. TRAP65 prevents system crash that occurs from unimplemented opcodes. TRAP65 can also be used to extend the 6502 instruction set by defining new instructions via software. \$69.95 (Not available for the C64.)
- d) Cursor scrolling package for scrolling Basic programs forward and reverse on the screen using the cursor up/down key. \$9.00 (Not available for the C64.)
- e) Standard Terminal Communications Package (STCP) consists of an RS232 interface board and RS232 cable. A sophisticated terminal software package is also included which provides constant display of time, alarm clock, upload/download from host computer, and many other features. The ultimate communications software package. Can be used with the Signalman modem, DC Hayes Smart Modem, or any RS232 type modem. STCP = \$129.95. Call for latest price on the Signalman modem and the DC Hayes Smart Modem.
- f) We sell many other items from EPROMs to printers. We actually use some of the brands we sell and can provide assistance for those devices.

23. REFERENCES

Books:

- Machine Language Programming for Beginners - Compute Publications
- 6502 Assembly Language Programming - Lance Leventhal
- Programming the PET/CBM - Raeto West
- CBM 64 Basic/O.S. Source Listing - DataCap of Belgium

MAE Macro Assembler/Text Editor for Commodore Computers

- 6502 Software Design
- Programming Manual for the 6502
- CBM Professional Computer Guide
- PET Personal Computer Guide
- The Whole PET Catalog
- Leo Scanlon
- Commodore Staff
- James & Ellen Strasma
- James & Ellen Strasma
- Strasmas & Beach

Magazines:

- Compute and Computes Gazette
- Commander
- Micro
- Midnight
- Torpet
- Plus all the valuable User Group Newsletters

```

0000      .LS                      ;LISTING 1 - UART DRIVER PROGRAM
0001
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 D8 "UART.M01" ;SET BAUD AND TABLE DELAYS

```

07F6 2368-2B5E 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 #$FF-MSK.IN ;      *
2009- 09 80      0080 ORA #MSK.OUT   ;      *
200B- 8D 43 E8 0090 STA PIA.DIR      ;      *
0100
200E- 20 47 20 0110 LP1      JSR GET.BIT
2011- D0 FB      0120 BNE LP1        ;BR. IF ALREADY SPACING
2013- AD 41 E8 0130 LP2      LDA PIA.PORT
2016- 29 40      0140 AND #MSK.IN
2018- F0 F9      0150 BEQ LP2        ;BR. IF MARKING
201A- A0 00      0160 LDY #00        ;CLEAR FOR DELAY FACTOR
201C- AD 41 E8 0170 LP3      LDA PIA.PORT ;GET BIT
201F- 29 40      0180 AND #MSK.IN ;      *
2021- F0 07      0190 BEQ GOT.COUNT
2023- C0 FF      0200 CPY #$FF

```

```

2025- F0 F5      0210      BEQ LP3
2027- C8          0220      INY
2028- D0 F2      0230 SKP.FF      BNE LP3
                0240
202A- 98          0250 GOT.COUNT TYA          ;MOVE COUNT TO R(A)
202B- A0 00      0260      LDY #00
202D- D9 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 BEEN SENT
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 DELO.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 ENO.5
                0580
2058- AC 01 20   0590 DLYFULL      LDY BIT.TIME
205B- B9 66 20   0600 ENO.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 *
2068- 00          0710      .BY 00          ;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 08         ;DELAY FULL FOR 9600 *
                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 *
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 D8 "UART.M02" ;UART OUTPUT DRIVER

```

0343 2368-26AB 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, 2 STOP
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

```

PAGE 04

```

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 #S7F          ;CLEAR BIT 7
20B5- C9 0D      0250      CMP #S0D          ;CR
20B7- F0 0B      0260      BEQ PAD.DEL
20B9- C9 0A      0270      CMP #S0A          ;LF
20BB- F0 07      0280      BEQ PAD.DEL
20BD- 60         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 #SFF-MSK.OUT
20D7- 90 02      0460      BCC SKP.BOUT
20D9- 09 80      0470      ORA #MSK.OUT
20DB- 8D 41 E8   0480 SKP.BOUT STA PIA.PORT
20DE- 60         0490      RTS
                0500
                0510
                0320      .FI D8 "UART.M03" ;UART INPUT DRIVER

```

0248 2368-25B0 UART.M03

```

                0010
                0020 ;      +++++ UART INPUT +++++
                0030
20DF- 08         0040 UART.IN  PHP
20E0- 78         0050      SEI
20E1- A9 00      0060      LDA #00                ;CLEAR CHAR.
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 20   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 DELO.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- 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

```

PAGE 05

20FD- 20 58 20	0230	JSR DLYFULL	;DELAY UNTIL MIDDLE OF NEXT
2100- 18	0240	CLC	
2101- 90 EE	0250	BCC LP.UI3	;LOOP 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	

END OF MAE PASS1

--- LABEL FILE: ---

BIT.OUT =20D2	BIT.TIME =2001	DELO.5 =204D
DLYFULL =2058	DONE.UI =2103	ENO.5 =205B
END.PGM =2109	EX.DEL =20D0	GET.BIT =2047
GOT.COUNT =202A	GOTBAUD =2035	LOOPDEL1 =2061
LOOPDEL2 =207A	LP.FI =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 =2000	NOT.THIS =2076	PAD.DEL =20C4
PAD.DELEN =20C8	PAD.DELX =20BE	PIA.DIR =E843
PIA.PORT =E841	SET.BAUD =2002	SKP.BOUT =20DB
SKP.FF =2028	SKP.UI1 =20F8	TBLBAUD =203F
UART.IN =20DF	UART.OUT =2097	UART.OUT1 =209E
UD.TBL1 =2066	UD.TBL2 =208C	
//0000,2109,2109		
}		
}		

2000-4FFC 1000-1FFC 7800
2368 1195

}

PAGE 01

```

0000      .LS                      ;LISTING 2 - SHELL-METZNER SORT
0001
0010 ;FILENAME:  GL.SORT.CTL
0020 ;      -----
0030      .CT
0040      .CE
0050
0060
0070
0080 ; *****
0090 ; *
0100 ; *          GENERAL LEDGER SORT          *
0110 ; *          -----                      *
0120 ; *
0130 ; *          COPYRIGHT 1980 BY C. MOSER OF EHS
0140 ; *          ALL RIGHTS RESERVED
0150 ; *
0160 ; *      WRITTEN:  29NOV1980
0170 ; *      UPDATED:
0180 ; *
0190 ; *
0200 ; *
0210 ; *
0220 ; *****
0230 ;
0240 ;THE FOLLOWING MACRO IS USED:
0250 ; MOVE CONTENTS OF Z-PAGE VARIABLE SOURCE TO
0260 ; Z-PAGE LOCATION DESTIN.
0270 ;
0280 !!!DSTA      .MD (SOURCE DESTIN)
0290      LDA *SOURCE
0300      STA *DESTIN
0310      LDA *SOURCE+1
0320      STA *DESTIN+1
0330      .ME
0340
0350 ;
0360      .PR "GENERAL LEDGER M.L. SORT"
0370      .PR "-----"
0380      .PR
0390
0400      .FI "GL.SORT.M00"          ;GET Z-PAGE VARIABLE ASS.

```

04DE 2633-2B11 GL.SORT.M00

```

0010 ;FILENAME:  GL.SORT.M00
0020 ;      -----
0030
0040 ;VARIABLES USED IN THE G.L. SORT:
0050 ;
0060      .RC                      ;TURN RELOCATOR OFF!!!!
0070      .BA $0045
0080 ZSTART
0045- 0090 R0      .DS 2
0047- 0100 R1      .DS 2
0049- 0110 R2      .DS 2
004B- 0120 R3      .DS 2

```

```

004D-      0130 R4      .DS 2
004F-      0140 R5      .DS 2
0051-      0150 R6      .DS 2
0053-      0160 R7      .DS 2
0055-      0170 R8      .DS 2
0057-      0180 R9      .DS 2
0059-      0190 R10     .DS 2
005B-      0200 R11     .DS 2
005D-      0210 R12     .DS 2
005F-      0220 R13     .DS 2
0061-      0230 R14     .DS 2
0063-      0240 R15     .DS 2
0065-      0250 R16     .DS 2
0067-      0260 R17     .DS 2
0069-      0270 R18     .DS 2
006B-      0280 R19     .DS 2
006D-      0290 R20     .DS 2
006F-      0300 R21     .DS 2
0071-      0310 R22     .DS 2
0073-      0320 R23     .DS 2
0075-      0330 R24     .DS 2
          0340
          0350      .RS      ;TURN RELOCATOR BACK ON!!!!
          0360
          0370 ZEND
          0380
          0390 ;NOTE:  LOCATIONS R0-R7 USED BY SM-SORT
          0400
          0410
          0420
          0430 ;OTHER LOCATIONS USED:
          0440
          0450 A1234      .DI R8      ;START OF ARRAY TABLE
          0460 A1         .DI R8      ;START OF ARRAY A1 HEADER
          0470 A2         .DI R9      ;          A2
          0480 A3         .DI R10     ;          A3
          0490 A4         .DI R11     ;          A4
          0500
          0510 SORT.A     .DI R12     ;START OF SORT ARRAY HEADER
          0520
          0530 ELEMN      .DI R13     ;ADDRS TO #,LO,HI
          0540 ELEMN      .DI R14     ;ADDRS TO #,LO,HI
          0550
          0560 STRXC      .DI R15+1   ;LENGTH OF STRING X
          0570 STRXAD     .DI R16     ;ADDRS OF STRING X
          0580 STRYC      .DI R17+1   ;LENGTH OF STRING Y
          0590 STRYAD     .DI R18     ;ADDRS OF STRING Y
          0600
          0610 N          .DI R19     ;N-TH ELEMENT
          0620 NAME       .DI R20     ;NAME OF ARRAY
          0630 PTR.A      .DI R21     ;POINTER TO CURRENT ARRAY
          0640
          0650 SORTLN     .DI R      ;# ELEMENTS (SAME AS SMSORT)
          0660
          0670 ST.ARRAY   .DE $002F   ;START OF ARRAY (BASIC)
          0680 END.ARRAY  .DE $0031   ;END OF ARRAY (BASIC)
          0690

```

PAGE 03

```

0700 SAVE.AREA .DE $033C          ;CASS #1 BUFFER
0710
0720
0730          .EN
0410
0420          .PR "INPUT BEGIN ASSEMBLY ADDRESS"
0430 BEG.ADDRS .IN BEG.ADDRS
0440          .BA BEG.ADDRS
0450
0460          .PR
0470          .PR "IS OBJECT CODE TO BE STORED."
0480          .PR "(1=YES, 0=NO)"
0490 OBJ       .IN OBJ
0500
0510          IFE OBJ-1
0520          .PR "INPUT ADDRESS TO STORE OBJECT CODE"
0530 STO.ADDRS .IN STO.ADDRS
0540          .MC STO.ADDRS
0550          .OS
0560          ***
0570
0580
0590          .FI "1:GL.SORT.M01"

```

1B49 2616-415F 1:GL.SORT.M01

```

0010 ;FILENAME: GL.SORT.M01
0020 ; -----
0030
0040
0050 ; <><> SORT AN$, EXCHANGE A1$,A2$,A3$,A4$ <><>
0060 ; -----
0070 ;THIS IS THE MAINLINE FOR THE GENERAL LEDGER SORT.
0080 ;BASIC SHOULD HAVE SET UP $0110-$0111 (LENGTH) AND
0090 ;$0112 (SORT ARRAY) AS FOLLOWS:
0100
0110 ;          POKE 272,E-INT(E/256)*256 : REM LO BYTE
0120 ;          POKE 273,INT(E/256)      : REM HI BYTE
0130 ;          POKE 274,INT(N)         : REM WHICH ARRAY - 1,2,3,4
0140 ;          SYS XXXXX               : REM -TRANSFER CONTROL TO SORT
0150 ;          IF PEEK(275)<>0 THEN PRINT "SORTING ERROR"
0160
0170 ; LOCATION $0113 = ERROR STATUS AS FOLLOWS:
0180 ; $00 (000) = NO ERROR
0190 ; $DD (221) = $0110, $0111 TOO LARGE
0200 ;          (EXCEEDS ARRAY DIMENSION)
0210 ; $FF (255) = ARRAY MISSING
0220
2000- 78      0230 SORT1      SEI          ;IRQ'S OFF FOR SPEED
2001- 20 06 20 0240          JSR SORT2    ;NOW SORT
2004- 58      0250          CLI          ;IRQ'S BACK ON
2005- 60      0260          RTS
0270
2006- 20 32 20 0280 SORT2      JSR SAVEZ    ;SAVE Z-PAGE
2009- A9 00      0290          LDA #00     ;DEFAULT TO NO SORT ERROR
200b- 8D 13 01 0300          STA $0113    ; *
200E- AD 10 01 0310          LDA $0110    ;SET UP LENGTH

```

```

2011- 85 45      0320      STA *SORTLN          ; *
2013- AD 11 01    0330      LDA $0111             ; *
2016- 85 46      0340      STA *SORTLN+1          ; *
2018- AD 12 01    0350      LDA $0112             ;GET WHICH ARRAY TO SORT ON
201B- 20 4C 20    0360      JSR FNDSORTA          ; *
201E- D0 0E      0370      BNE ERROR              ;BR. IF CANNOT FIND ARRAY
2020- 20 66 20    0380      JSR FNDA1234          ;FIND A1$-A4$
2023- D0 09      0390      BNE ERROR              ;BR. IF CANNOT FIND
2025- 20 AB 21    0400      JSR SMSORT            ;SORT VIA SHELL-METZNER
2028- D0 04      0410      BNE ERROR              ;BR. ON SORT ERROR
202A- 20 3F 20    0420      JSR RESTOREZ          ;RESTORE Z-PAGE
202D- 60         0430      RTS
202E- 8D 13 01    0440 ERROR STA $0113            ;INDICATE ERROR
2031- 60         0450      RTS
                0460
                0470
                0480
                0490 ; <><> SAVE ZERO PAGE VARIABLES <><>
                0500 ; -----
                0510 ;THIS MODULE SAVES THE PART OF ZERO PAGE THAT
                0520 ;IS USED BY THIS M.L. SORT.
                0530
2032- A2 00      0540 SAVEZ      LDX #00
2034- B5 45      0550 LP.SVZ    LDA *ZSTART,X
2036- 9D 3C 03    0560          STA SAVE.AREA,X
2039- E8         0570          INX
203A- E0 32      0580          CPX #ZEND-ZSTART
203C- 90 F6      0590          BCC LP.SVZ
203E- 60         0600          RTS
                0610
                0620
                0630 ; <><> RESTORE ZERO PAGE VARIABLES <><>
                0640 ; -----
                0650 ;THIS MODULE RESTORES THAT PART OF ZERO PAGE USED
                0660 ;BY THIS PROGRAM.
                0670
203F- A2 00      0680 RESTOREZ   LDX #00
2041- BD 3C 03    0690 LP.RESZ   LDA SAVE.AREA,X
2044- 95 45      0700          STA *ZSTART,X
2046- E8         0710          INX
2047- E0 32      0720          CPX #ZEND-ZSTART
2049- 90 F6      0730          BCC LP.RESZ
204B- 60         0740          RTS
                0750
                0760
                0770 ; <><> FIND ADDRESS OF THE SORT ARRAY <><>
                0780 ; -----
                0790 ;IF R(A)=1 THEN A1$ IS SORT ARRAY, IF -2 THEN A2$, ETC.
                0800
204C- 48         0810 FNDSORTA   PHA
204D- A9 41      0820          LDA #~A              ;FORM NAME
204F- 85 6D      0830          STA *NAME            ; *
2051- 68         0840          PLA                  ; *
2052- 09 B0      0850          ORA #SBO             ; *
2054- 85 6E      0860          STA *NAME+1          ; *
2056- 20 90 20    0870          JSR FIND.ARRAY      ;FIND ADDRESS
2059- D0 0A      0880          BNE ERR.FNDARY
                0890          DSTA (PTR.A SORT.A) ;PRESERVE ADDRS.

```

```

2063- A9 00      0900      LDA #00      ;R(A)=0 THEN NO ERROR
2065- 60          0910      ERR.FNDARY RTS
                0920
                0930
                0940 ; <><> FIND A1$, A2$, A3$, A4$ <><>
                0950 ; -----
                0960 ;SET UP ADDRESSES OF STRING ARRAYS A1$-A4$ IN
                0970 ;TABLE A1234.
                0980
2066- A2 00      0990      FNDA1234 LDX #00
2068- BD 88 20   1000      LPA1234 LDA STR.NAM,X      ;GET NAME FROM TABLE STR.NAM
                1010      STA *NAME      ; *
206D- BD 89 20   1020      LDA STR.NAM+1,X      ; *
                1030      STA *NAME+1      ; *
2072- 20 90 20   1040      JSR FIND.ARRAY      ;FIND ADDRESS
2075- D0 10      1050      BNE ERR.FNDA      ;BR. IF ERROR
2077- A5 6F      1060      LDA *PTR.A      ;PUT ADDRESS IN TABLE
2079- 95 55      1070      STA *A1234,X      ; *
207B- A5 70      1080      LDA *PTR.A+1      ; *
207D- 95 56      1090      STA *A1234+1,X      ; *
207F- E8         1100      INX      ; INC. FOR NEXT ARRAY
2080- E8         1110      INX      ; *
2081- E0 08      1120      CPX #08      ;STOP ON 4 NAMES
2083- 90 E3      1130      BCC LPA1234
2085- A9 00      1140      LDA #00      ;--LOOP-- UNTIL DONE
2087- 60         1150      ERR.FNDA RTS      ;R(A)=00 THEN NO ERROR
                1160
2088- 41 B1      1170      STR.NAM .BY 'A' $B1 ;A1$
208A- 41 B2      1180      .BY 'A' $B2 ;A2$
208C- 41 B3      1190      .BY 'A' $B3 ;A3$
208E- 41 B4      1200      .BY 'A' $B4 ;A4$
                1210
                1220
                1230 ; <><> FIND START ADDRESS OF AN ARRAY <><>
                1240 ; -----
                1250 ;THIS MODULE LOCATES THE START ADDRESS OF THE ARRAY
                1260 ;WHOSE NAME IS CONTAINED IN LOCATION NAME(2):
                1270 ;      INTEGER      FLOATING      STRING
                1280 ;      -----      -----      -----
                1290 ; NAME+0      ASC+128      ASC      ASC
                1300 ; NAME+1      ASC+128      ASC      ASC+128
                1310
2090- D8         1320      FIND.ARRAY CLD
                1330      DSTA (ST.ARRAY PTR.A) ;SET SEARCH AT START
2099- 38         1340      LOOPH SEC      ;CHECK FOR END OF ARRAY MEMORY
209A- A5 6F      1350      LDA *PTR.A+0      ; *
209C- E5 31      1360      SBC *END.ARRAY+0      ; *
209E- A5 70      1370      LDA *PTR.A+1      ; *
20A0- E5 32      1380      SBC *END.ARRAY+1      ; *
20A2- B0 24      1390      BCS NO.ENTY      ;BR. IF NOT IN THERE
                1400
20A4- A0 00      1410      LDY #00      ;TEST FOR NAME MATCH
20A6- B1 6F      1420      LDA (PTR.A),Y      ; *
20A8- C5 6D      1430      CMP *NAME+0      ; *
20AA- D0 07      1440      BNE NEXT.A      ; *
20AC- C8         1450      INY      ; *
20AD- B1 6F      1460      LDA (PTR.A),Y      ; *
20AF- C5 6E      1470      CMP *NAME+1      ; *

```



```

20B1- F0 18      1480      BEQ GOT.IT          ; *
                  1490
20B3- A0 02      1500 NEXT.A    LDY #02          ;GET MEMORY ALLOCATION
20B5- 18         1510          CLC              ; FOR ARRAY AND MOVE TO NEXT ARRAY
20B6- B1 6F      1520          LDA (PTR.A),Y     ; *
20B8- 65 6F      1530          ADC *PTR.A        ; *
20BA- 48         1540          PHA              ; *
20BB- C8         1550          INY              ; *
20BC- B1 6F      1560          LDA (PTR.A),Y     ; *
20BE- 65 70      1570          ADC *PTR.A+1     ; *
20C0- 85 70      1580          STA *PTR.A+1     ; *
20C2- 68         1590          PLA              ; *
20C3- 85 6F      1600          STA *PTR.A        ; *
20C5- 4C 99 20   1610          JMP LOOPM        ; *
                  1620
20C8- A9 FF      1630 NO.ENTY    LDA #$FF        ;R(A)=FF THEN NOT IN THERE
20CA- 60         1640          RTS
20CB- A9 00      1650 GOT.IT     LDA #00         ;R(A)=00 THEN GOT IT
20CD- 60         1660          RTS
                  1670
                  1680
1690 ; <><> COMPARE ELEMENTS X WITH Y <><>
1700 ; -----
1710 ;COMPARISON IS VIA: IF X>Y THEN C=T ELSE C=F
1720
1730 SMCMP      DSTA (SORT.A PTR.A) ;MOVE SORT.A TO PTR.A
1740          JSR SETUP.XY          ;GET X,Y'S #,LO,HI BLOCK
20D6- 20 FA 20   1750          BNE ERR.SMCMP
20D9- D0 1E      1760          LDY #00
20DB- A0 00      1770 LPCMP      CPY *STRYC      ;CHECK IF AT END STRING Y
20DD- C4 68      1780          BEQ X.GT.Y
20DF- F0 0D      1790          CPY *STRXC      ;CHECK IF AT END STRING X
20E1- C4 64      1800          BEQ X.LT.Y
20E3- F0 11      1810          LDA (STRXAD),Y    ;GET CHAR FROM STRING X AND
20E5- B1 65      1820          CMP (STRYAD),Y    ; COMPARE WITH Y
20E7- D1 69      1830          BNE EX.WC/S
20E9- D0 0C      1840 XXX        INY
20EB- C8         1850          BNE LPCMP
20EC- D0 EF      1860
20EE- C4 64      1870 X.GT.Y     CPY *STRXC      ;CHECK SO WE DON'T EXCHANGE
20F0- F0 04      1880          BEQ X.LT.Y        ; EQUAL ELEMENTS
20F2- 38         1890          SEC
20F3- A9 00      1900          LDA #00
20F5- 60         1910          RTS
20F6- 18         1920 X.LT.Y     CLC
20F7- A9 00      1930 EX.WC/S    LDA #00
20F9- 60         1940 ERR.SMCMP RTS
                  1950
                  1960
1970 ; <><> SET UP ELEMENTS X,Y <><>
1980 ; -----
1990 ;SET UP ELEMENTS X,Y IN STRXC/STRXAD, STRYC/STRYAD
2000 ;FOR ARRAY WHOSE ADDRESS IS IN PTR.A
2010
2020 SETUP.XY    DSTA (Y N) ;SET UP FOR ELEMENT Y FIRST
2102- 20 3F 21   2030          JSR STR.ELEM      ;RETURNS ADDR IN ELEMX
2105- D0 37      2040          BNE ERR.SETUP
                  2050          DSTA (ELEMX ELEM) ;PUT IN ELEM

```

PAGE 07

	2060		;
	2070		;NOW SET UP STRYC/STRYAD FOR ELEMENT Y
210F- A0 00	2080		LDY #00
2111- B1 61	2090		LDA (ELEM),Y
2113- 85 68	2100		STA *STRYC
2115- C8	2110		INY
2116- B1 61	2120		LDA (ELEM),Y
2118- 85 69	2130		STA *STRYAD
211A- C8	2140		INY
211B- B1 61	2150		LDA (ELEM),Y
211D- 85 6A	2160		STA *STRYAD+1
	2170		;
	2180		;NOW SET UP STRXC/STRXAD FOR ELEMENT X
	2190		DSTA (X N) ; SET UP FOR ELEMENT X
2127- 20 3F 21	2200		JSR STR.ELEM ;RETURNS ADDR IN ELEM
212A- D0 12	2210		BNE ERR.SETUP
	2220		;
	2230		;NOW SET UP STRXC/STRXAD FOR ELEMENT X
212C- A0 00	2240		LDY #00
212E- B1 5F	2250		LDA (ELEM),Y
2130- 85 64	2260		STA *STRXC
2132- C8	2270		INY
2133- B1 5F	2280		LDA (ELEM),Y
2135- 85 65	2290		STA *STRXAD
2137- C8	2300		INY
2138- B1 5F	2310		LDA (ELEM),Y
213A- 85 66	2320		STA *STRXAD+1
213C- A9 00	2330		LDA #00
213E- 60	2340	ERR.SETUP	RTS
	2350		
	2360		
	2370		; <><> FIND ADDRESS OF STRING ELEMENT N <><>
	2380		;
	2390		;FIND ADDRESS OF THE N-TH ELEMENT OF THE STRING
	2400		;ARRAY AT ADDRESS IN PTR.A.
	2410		;THE ELEMENTS ADDRESS WILL BE RETURNED IN ELEM(2)
213F- D8	2420	STR.ELEM	CLD
	2430		DSTA (PTR.A ELEM)
	2440		;
	2450		;DETERMINE IF DIM EXCEEDED
2148- A0 05	2460		LDY #05 ;POINT TO DIM QUANTITY
214A- 18	2470		CLC ;DIM-1-N
214B- B1 5F	2480		LDA (ELEM),Y ; *
214D- E5 6B	2490		SBC *N ; *
214F- C8	2500		INY ; *
2150- B1 5F	2510		LDA (ELEM),Y ; *
2152- E5 6C	2520		SBC *N+1 ; *
2154- 90 21	2530		BCC ERR.ELEM ;BR. IF DIM EXCEEDED
	2540		
2156- 18	2550		CLC
2157- A9 07	2560		LDA #07 ; *
2159- 65 5F	2570		ADC *ELEM ; *
215B- 85 5F	2580		STA *ELEM ; *
215D- 90 02	2590		BCC SK.PLUS7 ; *
215F- E6 60	2600		INC *ELEM+1 ; *
	2610	SK.PLUS7	
2161- 20 67 21	2620		JSR TM1 ;ELEMENT # * 3 (I.E. ADD 3 TIMES)
2164- 20 67 21	2630		JSR TM1

```

2167- 18      2640 TM1      CLC
2168- A5 6B      2650      LDA *N
216A- 65 5F      2660      ADC *ELEMx
216C- 85 5F      2670      STA *ELEMx
216E- A5 6C      2680      LDA #N+1
2170- 65 60      2690      ADC *ELEMx+1
2172- 85 60      2700      STA *ELEMx+1
2174- A9 00      2710      LDA #00
2176- 60      2720      RTS
2177- A9 DD      2730 ERR.ELEMN LDA #$DD      ;DIMENSION ERROR
2179- 60      2740      RTS
                2750
                2760
2770 ; <><> S-M EXCHANGE X,Y FOR A1$ - A4$ <><>
2780 ; -----
2790 ;THIS MODULE EXCHANGES THE X-TH AND Y-TH ELEMENTS
2800 ;OF EACH ARRAY A1$-A4$.
                2810
217A- A2 00      2820 SMEXCH      LDX #00
217C- B5 55      2830 LP.SMEXCH LDA *A1234,X      ;EXCHANGE X,Y FOR A1$ - A4$
217E- 85 6F      2840      STA *PTR.A+0      ; *
2180- E8      2850      INX      ; *
2181- B5 55      2860      LDA *A1234,X      ; *
2183- 85 70      2870      STA *PTR.A+1      ; *
2185- E8      2880      INX      ; *
2186- 8A      2890      TXA      ; *
2187- 48      2900      PHA      ; *
2188- 20 FA 20   2910      JSR SETUP.XY      ; *
218B- D0 0B      2920      BNE ERR.EXCH
218D- 20 99 21   2930      JSR EXCH.XY      ; *
2190- 68      2940      PLA      ; *
2191- AA      2950      TAX      ; *
2192- E0 08      2960      CPX #08      ; *
2194- 90 E6      2970      BCC LP.SMEXCH      ; --LOOP--
2196- A9 00      2980      LDA #00
2198- 60      2990 ERR.EXCH RTS
                3000
                3010
3020 ; <><> EXCHANGE STRING ELEMENTS X AND Y <><>
3030 ; -----
3040 ;STRING ELEMENTS X,Y ARE EXCHANGED BY THIS MODULE.
3050 ;ELEMx, ELEMx SHOULD CONTAIN ADDRESS OF ELEMENTS
3060 ;X,Y'S # LO HI BLOCK.
                3070
2199- A0 00      3080 EXCH.XY      LDY #00
219B- B1 5F      3090 LP.EXCH      LDA (ELEMx),Y
219D- 48      3100      PHA
219E- B1 61      3110      LDA (ELEMx),Y
21A0- 91 5F      3120      STA (ELEMx),Y
21A2- 68      3130      PLA
21A3- 91 61      3140      STA (ELEMx),Y
21A5- C8      3150      INY
21A6- C0 03      3160      CPY #03
21A8- 90 F1      3170      BCC LP.EXCH      ;--LOOP--
21AA- 60      3180      RTS
                3190
                3200      .EN
0600      .FI "1:SM-SORT.ASM"

```

OB93 2616-31A9 1:SM-SORT.ASM

```

0010 ;FILENAME: SM-SORT.ASM
0020 ;
0030
0040
0050 ; <><> SMSORT - SHELL METZNER SORT <><>
0060 ;
0070 ;THIS MODULE IS A GENERAL PURPOSE CORE SORT USING
0080 ;THE HIGH-SPEED SHELL-METZNER SORTING ALGORITHM.
0090 ;THE USER MUST PROVIDE THE FOLLOWING ADDITIONAL
0100 ;ROUTINES:
0110 ;
0120 ; SMCMP - MODULE WHICH COMPARES ELEMENTS X WITH Y
0130 ; VIA X>Y? IF TRUE THEN C=T ELSE C=F.
0140 ; Z=F IF SUBSCRIPT OVERFLOW ERROR.
0150 ;
0160 ; SMEXCH - MODULE WHICH EXCHANGES ELEMENTS X AND Y.
0170 ; Z=F IF SUBSCRIPT OVERFLOW ERROR.
0180 ;
0190 ;
0200 ;NOTES: 1-) X AND Y ARE 16-BIT MEMORY LOCATIONS.
0210 ; 2-) SMSORT CONSIDERS ELEMENT SUBSCRIPTS AS
0220 ; STARTING WITH 1,2,...,ETC., BUT ADJUSTS
0230 ; THESE IN X,Y FOR MODULES SMCMP AND SMEXCH
0240 ; AS 0,1,2,...,ETC.
0250 ;
0260 ;INPUT: # OF RECORDS TO BE SORTED MUST BE INITIALLY
0270 ; CONTAINED IN LOCATION R.
0280 ;
21AB- D8 0290 SMSORT CLD ;CLEAR DECIMAL MODE
0300 DSTA (R M) ; STORE R IN M (M=R)
0310 ;
21B4- 18 0320 LP.SM1 CLC ;M=INT(M/2)
21B5- 46 48 0330 LSR *M+1 ; *
21B7- 66 47 0340 ROR *M ; *
0350 ;
21B9- A5 48 0360 LDA *M+1 ;--EXIT-- IF M=0
21BB- 05 47 0370 ORA *M ; *
21BD- F0 63 0380 BEQ EX.SMSORT ; *
0390 ;
21BF- A9 01 0400 SK.NOTMO LDA #01 ;J=1
21C1- 85 49 0410 STA *J ; *
21C3- A9 00 0420 LDA #00 ; *
21C5- 85 4A 0430 STA *J+1 ; *
0440 ;
21C7- 38 0450 SEC ;K=R-M
21C8- A5 45 0460 LDA *R ; *
21CA- E5 47 0470 SBC *M ; *
21CC- 85 48 0480 STA *K ; *
21CE- A5 46 0490 LDA *R+1 ; *
21D0- E5 48 0500 SBC *M+1 ; *
21D2- 85 4C 0510 STA *K+1 ; *
0520 ;
0530 LP.SM2 DSTA (J H) ;H=J
0540 ;

```

```

21DC- 18      0550 LP.SM3      CLC                      ;V=H+M
21DD- A5 4D    0560      LDA *H                      ; *
21DF- 65 47    0570      ADC *M                      ; *
21E1- 85 4F    0580      STA *V                      ; *
21E3- A5 4E    0590      LDA *H+1                    ; *
21E5- 65 48    0600      ADC *H+1                    ; *
21E7- 85 50    0610      STA *V+1                    ; *
                0620      ;
21E9- 20 25 22 0630      JSR SUBSCRIPT                ;X=H-1, Y=V-1
                0640      ;
                0650      ;USER SUPPLYS SMCMP
21EC- 20 CE 20 0660      JSR SMCMP                    ;IF X>Y THEN C=T ELSE C=F
21EF- D0 33    0670      BNE ERR.SMSORT                ;BR. ON SUBSCRIPT ERROR
21F1- B0 13    0680      BCS H.GT.V                    ;BR. IF H>V
                0690      ;
                0700      ;H IS LESS THAN V
                0710 LP.SM4
21F3- E6 49    0720 H.LT.V      INC *J                ;J=J+1
21F5- D0 02    0730      BNE SK.INCJ                    ; *
21F7- E6 4A    0740      INC *J+1                      ; *
                0750      ;
21F9- 38      0760 SK.INCJ      SEC                      ;TEST IF J>K (K-J)
21FA- A5 4B    0770      LDA *K                      ; *
21FC- E5 49    0780      SBC *J                      ; *
21FE- A5 4C    0790      LDA *K+1                    ; *
2200- E5 4A    0800      SBC *J+1                      ; *
2202- B0 D0    0810      BCS LP.SM2                    ;BR. IF K>=J (J<=K)
2204- 90 AE    0820      BCC LP.SM1                    ;BR. IF K<J (J>K)
                0830      ;
                0840      ;H IS GREATER THAN V
                0850      ;NOTE: SUBSCRIPTS ALREADY SET UP VIA SMCMP.
                0860      ;/\USER SUPPLYS SMEXCH/\
2206- 20 7A 21 0870 H.GT.V      JSR SMEXCH                ;EXCHANGE H AND V
2209- D0 19    0880      BNE ERR.SMSORT                ;BR. IS SUBSCRIPT ERROR
                0890      ;
220B- 38      0900      SEC                      ;H=H-M
220C- A5 4D    0910      LDA *H                      ; *
220E- E5 47    0920      SBC *H                      ; *
2210- 85 4D    0930      STA *H                      ; *
2212- A5 4E    0940      LDA *H+1                    ; *
2214- E5 48    0950      SBC *H+1                    ; *
2216- 85 4E    0960      STA *H+1                    ; *
                0970      ;
2218- A5 4E    0980      LDA *H+1                    ;BR. TO LP.SM4 IF H<1
221A- 30 D7    0990      BMI LP.SM4                    ;ELSE BR. TO LP.SM3
221C- 05 4D    1000      ORA *H                      ; *
221E- F0 D3    1010      BEQ LP.SM4                    ; *
2220- D0 BA    1020      BNE LP.SM3                    ; *
                1030      ;
2222- A9 00    1040 EX.SMSORT  LDA #00
2224- 60      1050 ERR.SMSORT RTS
                1060      ;
                1070      ;
                1080      ;<><> SETUP SUBSCRIPTS X,Y <><>
                1090      ;-----
                1100      ;
                1110      ;THIS MODULE SETS UP X,Y AS SUBSCRIPTS FOR
                1120      ;MODULES SMCMP AS X=H-1 AND Y=V-1. THE

```

```

1130 ;REASON IS THAT SMSORT TREATS SUBSCRIPTS AS
1140 ;1,2,...N AND SMCMP AND SMEXCH USE 0,1,...N.
1150
2225- 38      1160 SUBSCRIPT SEC                ; X=H-1
2226- A5 4D    1170      LDA *H                ; *
2228- E9 01    1180      SBC #01                ; *
222A- 85 51    1190      STA *X                ; *
222C- A5 4E    1200      LDA *H+1              ; *
222E- E9 00    1210      SBC #00                ; *
2230- 85 52    1220      STA *X+1              ; *
                1230 ;
2232- 38      1240      SEC                    ; Y=V-1
2233- A5 4F    1250      LDA *V                ; *
2235- E9 01    1260      SBC #01                ; *
2237- 85 53    1270      STA *Y                ; *
2239- A5 50    1280      LDA *V+1              ; *
223B- E9 00    1290      SBC #00                ; *
223D- 85 54    1300      STA *Y+1              ; *
223F- 60      1310      RTS
                1320
                1330
                1340
1350 ;SM-SORT VARIABLES:
1360 ;-----
1370 R          .DI R0                        ;# OF RECORDS TO BE SORTED
1380 M          .DI R1
1390 J          .DI R2
1400 K          .DI R3
1410 H          .DI R4
1420 V          .DI R5
1430
1440 ;VARIABLES PASSED TO SMCMP AND SMEXCH:
1450 ;-----
1460 X          .DI R6
1470 Y          .DI R7
1480
1490
1500          .EN
0610
0620 END.PGM  .EN

```

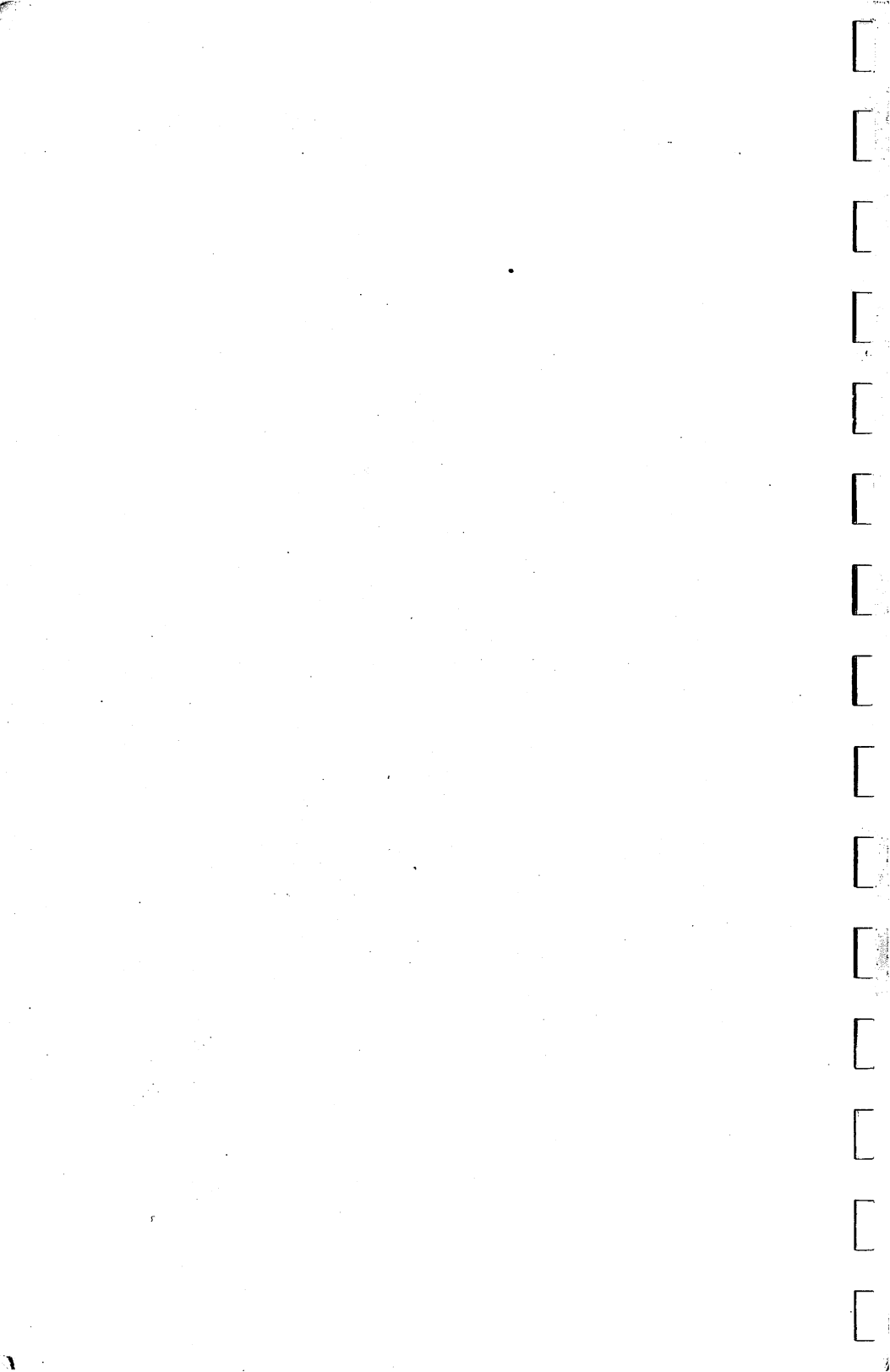
END OF MAE PASS!

--- LABEL FILE: ---

A1 =0055	A1234 =0055	A2 =0057
A3 =0059	A4 =005B	BEG.ADDRS =2000
DESTIN =004D	ELEMX =005F	ELEMV =0061
END.ARRAY =0031	END.PGM =2240	ERR.ELEMN =2177
ERR.EXCH =2198	ERR.FNDA =2087	ERR.FNDARY =2065
ERR.SETUP =213E	ERR.SMCMP =20F9	ERR.SMSORT =2224
ERROR =202E	EX.SMSORT =2222	EX.WC/S =20F7

PAGE 12

EXCH.XY -2199	FIND.ARRAY -2090	FNDA1234 -2066
FNDSORTA -204C	GOT.IT -20CB	H -004D
H.GT.V -2206	H.LT.V -21F3	J -0049
K -004B	LOOPM -2099	LP.EXCH -219B
LP.RESZ -2041	LP.SM1 -21B4	LP.SM2 -21D4
LP.SM3 -21DC	LP.SM4 -21F3	LP.SMEXCH -217C
LP.SVZ -2034	LPA1234 -2068	LPCMP -20DD
M -0047	N -006B	NAME -006D
NEXT.A -20B3	NO.ENTY -20C8	OBJ -0000
PTR.A -006F	R -0045	RO -0045
R1 -0047	R10 -0059	R11 -005B
R12 -005D	R13 -005F	R14 -0061
R15 -0063	R16 -0065	R17 -0067
R18 -0069	R19 -006B	R2 -0049
R20 -006D	R21 -006F	R22 -0071
R23 -0073	R24 -0075	R3 -004B
R4 -004D	R5 -004F	R6 -0051
R7 -0053	R8 -0055	R9 -0057
RESTOREZ -203F	SAVE.AREA -033C	SAVEZ -2032
SETUP.XY -20FA	SK.INCJ -21F9	SK.NOTMO -21BF
SK.PLUS7 -2161	SMCMP -20CE	SMEXCH -217A
SMSORT -21AB	SORT.A -005D	SORT1 -2000
SORT2 -2006	SORTLN -0045	SOURCE -0049
ST.ARRAY -002F	STR.ELEM -213F	STR.NAM -2088
STRXAD -0065	STRXC -0064	STRYAD -0069
STRYC -0068	SUBSCRIPT -2225	TM1 -2167
V -004F	X -0051	X.GT.Y -20EE
X.LT.Y -20F6	XXX -20EB	Y -0053
ZEND -0077	ZSTART -0045	
//0000,2240,2240		
]		
]		
2000-4FFC 1000-1FFC 7800		
2616 1365		
]		



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

