GETTING STARTED WITH OSS

CONGRATULATIONS !!!

You have purchased what we believe is by far the most advanced software development package available for the Atari 800 and Atari 400 personal computers.

This package will run on any Atari 800 or Atari 400 with at least 32K bytes of RAM. Since no OSS software uses any routines in any cartridge, you may fully utilize all the RAM in even a 48K byte Atari.

CAUTION: If you have ANY cartridge plugged into your Atari, you will not be able to utilize more than 40K bytes of memory. This a hardware feature of the Atari and can not be changed via software. If you need the power of 48K bytes of RAM, REMOVE ALL CARTRIDGES.

There are, however, some circumstances under which you may need a cartridge for your own development work. OSS CP/A is completely compatible with all known Atari cartridges.

HOW TO USE YOUR OSS PACKAGE

1. Check the contents of your package. If you ordered just BASIC A+, there should be a BASIC A+ manual (an addendum to the Atari Basic manual). If you ordered CP/A, there should be a CP/A manual and an EASMD (Editor/ASseMbler/Debug) manual.

2. There should be a license agreement. FILL THIS OUT NOW AND SEND IT TO US ! Aside from its obvious purpose, the agreement is YOUR ticket to ***SUPPORT***. Yes, we do answer phone questions. Yes, we do respond to bugs. BUT ONLY for those persons who send back their license !!!

3. Turn on your disk drive(s) and screen, leave the Atari computer off. If you purchased CP/A, place the CP/A disk in drive 1. If you purchased only BASIC A+, place an Atari DOS master disk in drive 1. Boot up the system by turning on the computer's power. That's all there is to it! Follow the manual directions for running the program you desire.
   Note: special instructions for running BASIC A+ under Atari DOS are in the beginning of the BASIC A+ manual.

4. We strongly urge you to immediately make a backup copy of your OSS diskette. You may do this using DUPDSK (see CP/A manual). [Or use the Atari DUPLICATE DISK menu DOS command.]

5. Sit back and enjoy the power of a REAL computer system.

OPTIMIZED SYSTEMS SOFTWARE

OSS EASMD

for the Atari 800 and Atari 400

MAY 1981

Version 1.1

NOTICE

OPTIMIZED SYSTEMS SOFTWARE reserves the right to make changes or improvements in the product decribed in this manual at any time and without notice.

Optimized Systems Software
10379 Lansdale Ave.
Cupertino CA. 95014

(408) 446-3099

TABLE OF CONTENTS

START UP

Editor/Assembler/Debug (EASMD)

FOR START UP:

Put the OSS diskette in disk drive 1 and turn on the power.

This will load the Operating System and execute OS/A+. Now enter:

EASMD (return)

This will load the Editor/Assembler/Debug and start executing it. See the OS/A+ manual for other capabilities.

WARMSTART:

The user can return to OS/A+ using the EASMD command CP or by using the SYSTEM RESET key. He can then re-enter EASMD by using the OS/A+ command RUN (if he has not loaded another program). This does a warm start which preserves text lines already in memory.

BACK-UP COPY:

On a dual drive system, simply use COPY or DUPDSK. On a single drive system, one can use DUPDSK or one can make a back-up copy of EASMD on another diskette via the OS/A+ SAVE command.

System RAM size	32k	40k	48k
Start address	5700	7700	9700
End address	7C00	9C00	BC00

File Name: EASMD.COM (or any .COM name of your choice)

NOTE: For a full explanation of OS/A+ commands see the OS/A+ reference manual.

SYNTAX CONVENTIONS

The following conventions are used in the discussion of syntax in this manual.

1) Capital letters denote commands, etc. which must be typed by the user exactly as shown. (eg. LIST, DEL)

2) Lower case letters denote types of items which may be used. The various types are shown in the next section. (eg. lno)

3) Items in square brackets are optional (eg. [,lno])

4) Multiple items in braces indicate that any one may be used. (eg. {A})
{Q}

TYPES OF ITEMS:

The following types of items are used in describing syntax commands.

lno — line number (in range 0 to 65535).

string — A string of ASCII characters.

adr — A memory address (given in hex).

data — A list of hexadecimal values separated by commas.

Example: AB,12,FE

incr — Increment a decimal value.

filespec — See Atari DOS manual or OS/A+ reference manual for full format.
Generally you may use

D[n]:xxxxxxxx.yyy	for disk files
P:	for the printer
etc.	

Note that in EASMD filespecs must ALWAYS be prefaced with a pound sign (#).

EDITOR

The Editor allows the user to enter and edit lines of ASCII text.

TEXT FORMAT

Lines of ASCII text received by the Editor are stored in memory. A line consists of a line number (0 to 65535), text information and a carriage return. The text information that is between the line number and the carriage return is stored exactly as it is received. Thus any combination of ASCII data is valid text.

Example: 1000LITTLE GREEN APPLES

This is valid text as far as the Editor is concerned.

NOTE: The Assembler, however, expects a blank after the line number and will not look at the first character after the line number. Thus

```
1000ABC     LDA    #0
```

is seen as

```
1000 BC     LDA    #0
```

Example: 100 PRINT X<SIN(X)

The Editor can be used to create and edit Basic programs.

TABLES

The text area and other user tables are built starting at an address in low memory and growing towards high memory. The user can change this address using the LOMEM command.

The user can also change the highest address the Editor will use for user text by using the change memory command in the Debug monitor to change UHIMEM. (See memory map for UHIMEM address).

COMMAND FORMAT

The stored lines of text are manipulated by Editor commands. A command is distinguished from text by the absence of a line number. Any line of data received by the program that does not begin with an ASCII numeric is considered to be a command. The Editor will examine the characters to determine what function to perform. If these characters do not form a valid command, or if the command syntax is invalid, the Editor will respond with:

WHAT?

LINE PROMPTING

The Editor will prompt the user each time a command has finished executing by printing:

EDIT

The cursor will appear on the following line. Since some commands take awhile to execute, the prompt serves to tell the user when more input is allowed.

EDITOR COMMAND SYNTAX AND DECRIPTION

NEW

NEW will delete all user text from the text area in memory.

```
DEL     lno
DEL     lno1,lno2
```

DEL deletes the specified line number (lno) or all the lines in the range lno1 through lno2.

```
FIND    /string/
FIND    /string/,A
FIND    /string/lno1[,lno2]
FIND    /string/lno1[,lno2],A
```

The FIND command will search the specified lines (all or lno1 through lno2) for the "string" between the specified delimiters. The delimiters may be any character other than blank. The second delimiter must be the same as the first.

If "A" is specified, any line that contains a matching string will be printed at the user terminal. If "A" is not specified, then only the first line that contains a matching string will be printed.

```
LIST
LIST    #filespec
LIST    lno1[,lno2]
LIST    #filespec,lno1[,lno2]
```

The LIST command will cause all lines in the specified range to be listed to the screen (or to a device/file when "#filespec" is specified).

If "lno1" is less than the line number of the first text line, then listing will start with the first line. If "lno2" is greater than the line number of the last text line, then listing will end with the last line.

Hitting the break key will stop the LIST.

Example: LIST #D1:EX.TST

Will list all lines to a file EX.TST on drive 1.

Example: LIST #P:

Will list to the printer.

```
PRINT
PRINT    #filespec
PRINT    lno1[,lno2]
PRINT    #filespec,lno1[,lno2]
```

Print is exactly the same as LIST except that the line numbers are not PRINTed, and that the EDIT ready prompt will not be printed after the last line until the user hits the RETURN key.

```
ENTER    #filespec[,M]
```

The ENTER command causes previously LISTed text from the device or file specified by #filespec to be re-entered. The optional "M" parameter specifies that the new text is to be merged with the text currently in memory. If "M" is not present, then the text area will be cleared before starting the ENTER.

Example: ENTER #D2:XXX
Will re-enter the text that was listed to the file XXX on drive 2.

```
NUM
NUM      slno,incr
NUM      incr
```

The number command is used to automatically attach line numbers to user lines. The user is prompted with the next line number. A blank automatically follows the line number. The "slno" parameter specifies the starting line number. The "incr" parameter is the line number increment.

The default "incr" is 10. The default "slno" is the last text line number plus "incr".

Hitting RETURN after the line number prompt terminates NUMber mode.

```
REN
REN      slno,incr
REN      incr
```

The REN command renumbers the text. The first line number will be "slno". The line numbers will increment by incr. The default "slno" and "incr" is 10.

```
REP      /old string/new string/
REP      /old string/new string/,{A}
                                 {Q}
REP      /old string/new string/lno1[,lno2]
REP      /old string/new string/lno1[,lon2],{A}
                                           {Q}
```

The REP command will search the specified lines (all or lno1 through lno2) for the "old string" (between specified delimiters). The delimiters follow the same

rules as the delimiters for FIND.

The "A" option causes all occurrence of "old string" to be replaced with "new string" (between the same specified delimiters).

If the "Q" option is specified then when each match is found, the line is listed and the user is allowed to specify change (Y followed by RETURN) or don't change (RETURN alone) this occurrence. Hitting BREAK will terminate the REPlace and return to the Editor.

If neither "A" or "Q" is specified, only the first occurrence of "old string" will be replaced with "new string".

NOTE: Each time a replace is done the changed line is listed.

SIZE

The SIZE command prints the users low memory address, the highest used memory address, and the highest usable memory address (UHIMEM).

LOMEM adr

LOMEM command changes the address at which user tables start.

NOTE: The LOMEM command will destroy any user statements in memory.

NOTE: This command can be used to reserve a space between the default low memory and the new low memory address. This space can then be used for the object output from the assembler.

CP
DOS

CP or DOS returns to the OSS Control Program (OS/A+)

BYE

BYE returns to the Atari Memo Pad.

ASM
ASM [#filespec1], [#filespec2], [#filespec3]

The ASM command assembles source code and produces object code and a listing.

By default:

1) The source "device" is the user text area.
2) The listing "device" is the screen.
3) The object "device" is memory.

These defaults can be overridden as follows:

filespec1 - source code file or device
filespec2 - listing file or device
filespec3 - object file or device

A "filespec" can be omitted by substituting a comma. in which case the default holds for that parameter.

Example: ASM #D1:SOURCE,#D2:LIST,#D1:OBJ

In this example, the source will come from D1:SOURCE, the listing will be written to D2:LIST, and the object will be written to D1:OBJ.

Example: ASM ,,#D3:OBJ

In this example the source will come from user text area in memory, the listing will go to the screen, and the object code will be written to the file OBJ on disk drive 3.

Example: ASM ,#P:

In this example the listing will go to the printer.

NOTE: See the .OPTion directive for full information. about when object is actually written to the specified file (or memory).

BUG

The BUG command causes the debug monitor to be entered.

DEBUG

The Debug Monitor allows the user to perform controlled execution of machine code, examine memory, alter memory, move memory blocks and verify the equality of memory blocks.

COMMAND FORMAT

The Debug Monitor assumes that any line of data that it receives is a command. If the data does not form a valid command, the Debug Monitor responds with:

WHAT?

LINE PROMPTING

The Debug Monitor will signal completion of a command by printing:

DEBUG

The cursor will appear on the following line.

NOTE: If the user is getting a syntax error indication (WHAT?) on what he thinks is a valid command, he should check the prompt message (DEBUG/EDIT) to verify that he is in the correct mode.

DEBUG COMMAND SYNTAX AND DESCRIPTION

G [adr]

The G Command (Go) transfers control to the specified address via a JMP command. If "adr" is not specified, then the current monitor program counter is used.

T [adr]

The T Command (Trace) causes instructions to be executed starting at "adr". If "adr" is not specified then the current monitor program counter is used. As each instruction is executed, its address, mnemonic and operand will be displayed along with the current values in the 6502 A, X, Y, P(status), & S(stack) registers.

Hitting the break key (BREAK) will terminate trace.

S [adr]

The S Command (Step) is exactly like the T command except that only one instruction is executed.

D

D adr1[,adr2]

The D command (Display Memory) will cause memory from "adr1" to "adr2" to be displayed in hexadecimal. If "adr2" is omitted, then 8 bytes are displayed (ie, adr2 = adr1 + 8).
If "adr1" is omitted, then this display starts where the last display left off (ie, at the last "adr2" + 1).

Hitting the break key (BREAK) will terminate Display.

C [adr1]<data

The C command (Change Memory) is used to alter memory starting at "adr". If "adr" is not specified, then Change uses the most recent "adr1" if D was the last command, or the next unchanged address if C was the last command.
The "data" is a list of 1 byte hex values separated by commas.

Example: C 5000<3,CD,1F

Will change locations 5000 thru 5004 to 3,CD,1F,2,3 respectively.

Multiple commas may be used to skip over memory addresses without changing the contents to reach the desired address.

Example: C 5000<3,,1F

will change hex location 5000 to 3, location 5002 to 1F, and location 5001 will be unchanged.

L

L adr1[,adr2]

The L command (list) will cause the instructions located at "adr1" to be disassembled and displayed with the address, instruction mnemonic and operand. If "adr2" is not specified, then twenty instructions will be listed. If the address field ("adr1") is not specified, then this list will start where the last one left off.

Hitting the break key (BREAK) will stop the listing.

M tadr<fsadr,feadr

The M command (Move) moves data from the address "fsadr" through the address "feadr" to the address specified with "tadr".

tadr - "move to" address
fsadr - "move from" start address
feadr - "move from" end address

V adr1<adr2,adr3

The V Command (Verify) compares the memory starting at "adr1" with the memory located at "adr2" through "adr3". If any of the compared bytes mismatch, then address and data bytes will be displayed.

DR

The DR command (Display Registers) will cause the A,X,Y, status (P) and stack (S) registers to be displayed in hexidecimal.

CR <data

The CR Command (Change Registers) is used to change the registers. Registers are assumed to be in the order: A,X,Y, status (P) stack (S), so that the first byte of data goes into A register the second into X, etc.

As in the C command, "data" is a list of hexadecimal values separated by commas and field may be skipped by use of multiple commas.

Example: CR<FF,,3

will set A=FF and Y=3. It will leave X,P and S unchanged.

X

The X command (exit) will cause control to return to the Editor.

A

The A command (Assemble) will cause the system to enter into the Debug Assembler mode. No prompt other than the cursor is used in this mode.

The Debug Assembler is a line-at-a-time assembler that uses 6502 mnemonics and operand format. Relative branch operands are specified as the actual "branch to" address; the Assembler creates the relative address.

The format of each line is:

[adr]< assembler code

The Debug Assembler keeps track of the location counter so that if "adr" is omitted, the next consecutive address is used.

Entering only a carriage return will return the user to the Debug monitor.

Example: While in Debug mode the user enters:

```
A
5000< LDA#3
< BNE $5010
```

The "A" puts the user into the Debug Assembler. The next two statements will cause memory to contain the following:

```
5000  A9 03
5002  D0 0C
```

NOTE: The blank after the "<" is required.

NOTE: The Debug Assembler accepts both decimal and hex numbers as operands; therefore, hex operands must be preceeded by "$".

BREAK POINTS

BRK instructions must be individually set and removed by the user.

Step and Trace intercept the BRK instruction and simulate its execution.

ASSEMBLER

The Assembler gets control when ASM is typed into the Editor. For the ASM command syntax, see the Editor section.

Hitting the break key (BREAK) will stop the assembly.

ASSEMBLER INPUT

Input to the Assembler is lines of ASCII data as entered into the Editor. Source lines are of the form:

(line number) (blank) (source statement)

where source statement is of the form:

[label] {6502 instruction}
{ directive }

A source statement may consist of a label only, or it may be blank.

In general the format is as specified in the MOS Technology 6502 Programming Manual. We recommend that the user unfamiliar with 6502 assembly language programming should purchase:

"Programming the 6502" by Rodney Zaks
or
"6502 Assembly Language Programming" by Lance Leventhal.

INSTRUCTION FORMAT:

A) Instruction mnemonics as described in the MOS Technology 6502 Programming Manual.

B) Immediate operands begin with #

C) "(Operand,X)" and "(Operand),Y" for indirect addressing.

D) "Operand,X" and "Operand,Y" for indexed addressing.

E) Zero page and forward equates recognized and evaluated within the limits of a two pass assembler.

F) "*" refers to the location counter.

G) Comment lines begin with ";"

H) Hex constants begin with "$"

I) The "A" operand is reserved for accumulator addressing.

DIRECTIVES

.TITLE "string"

The .TITLE directive allows the user to specify a title to be used in conjunction with .PAGE

.PAGE ["string"]

The .PAGE directive allows the user to specify a page heading. It issues an ASCII form feed (hex OC) and prints the most recent title and page headings.

NOTE: The most recent title and page headings are also printed every time 52 lines of source code have been assembled.

.BYTE expression and/or "string" list

The .BYTE directive sets a one byte value for each expression and the ASCII equivalent of each character of each string into the object code.

Example: .BYTE 3,"ABC",7,"X"

produces:

03 41 42 43 07 58

.WORD expresion list

The .WORD directive sets a two byte value into the object code for each expression in the list. The value is in 6502 address order (least significant byte, most significant byte).

Example: .WORD $1000,$2000

produces:

00 10 00 20

.DBYTE expression list

The .DBYTE directive sets a two bute value into the object code for each expression in the list. The value is in most significant, least significant byte order.

Example: .DBYTE $1000,$2000

produces:

.TAB expression,expression,expression

The .TAB directive sets displacements for the printing of the op code, operand, and comment fields of the source line. Each expression is a one byte value.
Defaults are 12, 17, 27 .

.OPT assembler option list

The .OPT directive allows the user to specify certain options affecting the assembly.

Possible options are :

```
LIST/NOLIST
NOOBJ/OBJ
ERR/NOERR
EJECT/NOEJECT
```

LIST/NOLIST	determines if a listing is produced.
NOOBJ/OBJ	determines if object code is produced.
ERR/NOERR	determines if error messages are printed.
EJECT/NOEJECT	determines if a form feed, title, and page are printed after 52 source lines.

Defaults are:

```
OBJ - when the object is going to a device/file.
NOOBJ - when the object "device" is memory.
LIST, ERR, EJECT  - in all cases.
```

*= expression

The *= directive serves the function of ORG. It sets the current location counter for subsequent source statements.

NOTE: *= must be written with no intervening blanks.

= expression

The = directive is an equate (EQU) statement. It must always be written:

```
LABEL  = expression
```

The value of the "expression" is assigned to "LABEL".

. IF expression , label

The . IF statement allows limited conditional assembly.
If the "expression" is true (non-zero), the Assembler skips all following lines up to the one that begins with the "label". If the "expression" is false (zero), assembly continues normally.

NOTE: There can be NO blank between the comma and label.

. INCLUDE #filespec

The . INCLUDE directive allows source code from the device or file specified in "filespec" to be inserted into the assembly.

NOTE: . INCLUDE's can not be nested. That is, a file that was included cannot contain a . INCLUDE directive.

NOTE: . INCLUDE cannot be the last statement. It must be followed by a . END or some other statement.

. END

The . END directive terminates the assembly.

EXPRESSIONS

Expressions are evaluated strictly left to right. Parentheses are not valid. Valid operators are:

\+ - * / & (& is a binary AND)

These are all binary operands. ("-5 + 3" is not valid, but "0 - 5 + 3" is valid.)

Example: LDX # ADDR/256
LDY # ADDR&255
Will put the MSB and LSB portions of the address of "ADDR" into X and Y respectively.

STRINGS:

Strings must be enclosed in double quotes:

. BYTE "THIS IS A MESSAGE"

The single character representation for the immediate operand :

#'C

LABEL:

Labels must start in the 1st colunm after (line number)(blank). A label may consist of up to 255 characters. It must start with an alpha character and may be followed by alpha-numeric characters or the character "." .

NOTE: The character "A" by itself can not be a label.

COMMENTS:

Comment lines start with the character ";"

No special character is needed to delineate a comment after the assember code on a line. When the assember recognizes the end of the operand field (or op code field for instructions without operands), the rest of the line is assumed to be comment.

NOTE: This can give unexpected results in some cases.

Example:

```
LDA  7A    GET NUM
```

will generate

A5 07

The decimal number "7" is terminated by the character "A". The comment in this case is:

```
A     GET NUM
```

If the user wishes to specify the hex location 7A, he must use $7A.

ERROR DESCRIPTION

When an error occurs the system will print out:

ERROR- XX [message]

Where XX represents an error number. When the Assembler finds more than 1 error in a line, up to 3 error numbers will be listed. Most ERRORs will produce a message (similar to those below).

ERROR NUMBERS

1 - MEMORY FULL

All available memory has been used. If issued from Editor, no more statements can be entered. If issued by the Assembler, no more labels can be defined.

2 - INVALID DELETE RANGE

The first number specified in a delete range does not exist.

3 - DEBUG ASSEMBLER ADDRESS ERROR

The origin address on an input line to the Debug Assembler is incorrectly specified.

4 - BLANK REQUIRED AFTER LINE NUMBER

The Assembler expects the first character after a line number to be a blank. The first character was ignored.

5 - UNDEFINED REFERENCE

Assembler has encountered an undefined label.

6 - ASSEMBLER SYNTAX ERROR

7 - DUPLICATE LABEL

The Assembler has encountered a label that is already defined.

8 - BUFFER OVERFLOW

An internal buffer is full. Try making the source code shorter.

9 - EQUATE HAS NO LABEL

An equate (=) must have a label.

10 - VALUE OF EXPRESSION > 255

The value of an expression was greater than 255 but a one byte value was required.

11 - NULL STRING

A null string is invalid in .BYTE

12 - INVALID ADDRESS OR ADDRESS TYPE

An invalid address type was specified for the mnemonic.

13 - PHASE ERROR

The address generated for a label in pass 2 of the Assembler is different from the address generated by pass 1. Other errors can also cause this error to be generated.

14 - UNDEFINED/FORWARD REFERENCE FOR *= (ORG)

The operand for the *= directive must already be defined when the directive is encountered. A forward reference on an *= directive is invalid.

```
Example:        1000   *=ABC
                2000 ABC = $1000
            Will produce this error.
```

15 - LINE TOO LONG

The input line is too long. (This error results when there are too many distinct items on a line for the syntax processor to handle.) Break the input line into multiple lines.

16 - INVALID INPUT LINE

The Assembler received a line that does not start with a valid line number.

17 - LINE NUMBER TOO BIG

The line number on an Editor input line is too big. (greater than 65535).

19 - NO ORIGIN (*=) SPECIFIED
Either no origin (*=) was given or it was specified as 0. This error will cause the assembly to terminate.

20 - OVERFLOW ON NUM OR REN

On NUM or REN command the line number generated went over 65535. If REN caused this error, the line numbers are now invalid. Issuing a valid REN command will correct the problem.

21 - NESTED INCLUDE INVALID

An INCLUDEd file can not contain a .INCLUDE directive.

NOTES

LOMEM/HIMEM:

A default low memory address is set when the system is booted up. EASMD does NOT automatically reset this value.
If a program (for example, a device handler) sets lomem and then EASMD is entered, this address remains unchanged.

EASMD does set a default UHIMEM (highest usable memory for EASMD tables, including user text) which can be changed by using the Change memory command in the Debug monitor.

IOCBs USED:

No command in the Debug monitor does I/O to a device other than the screen or keyboard; therefore, IOCBs 1 through 7 are not used by the system itself while in Debug mode.

Several commands in the Editor however, can do I/O to other devices (ENTER, ASM, etc). In these cases, the Editor must use one or more IOCBs. (The Editor uses IOCBs 1 through 4). Unpredictable things can happen to a file that was allocated to one of these IOCBs and never closed. The user who is debugging code that does I/O needs to be aware of this fact.

ALWAYS CLOSE FILES.

Note that returning to OS/A+ will ALWAYS cause all files to be closed.

LOAD/SAVE:

To load and save code for debugging, use the OS/A+ LOAD and SAVE command. To return to EASMD after LOADing a file, the user must enter RUN followed by the coldstart or warmstart address (see memory map). This will work if the user's code did not overlay any memory used by EASMD.

NUMBERS:

The Editor/Assembler/Debug (EASMD) uses positive integers and hex numbers, but it uses a Floating Point package for ASCII to integer conversion. This can give some unexpected results.

Example: LDA #6.7

produces

A9 07

Example: 100. 100.1 99.9

entered as line numbers each produces the line number 100.

BASIC:

The Editor can be used to create and edit OSS BASIC A+ programs. Of course, the user must take care of changing line numbers in GOTO, GOSUB, etc. whenever RENumber is used.

MEMORY MAP

The following are some memory addresses used by EASMD which may be of interest to the user. All addresses are given in hex.

size of RAM	32K	40K	48K
zero page free for user	B0-CF	B0-CF	B0-CF
user high memory (UHIMEM)	0498	0498	0498
Coldstart	5700	7700	9700
Warmstart	5703	7703	9703

SYNTAX SUMMARY

EDITOR

```
ASM
ASM       [#source filespec], [#list filespec], [#object filespec]

BUG

BYE

CP

DEL       lno
DEL       lno1,lno2

DOS

ENTER     #filespec

FIND      /string/
FIND      /string/,A
FIND      /string/lno1[,lno2]
FIND      /string/lno1[,lno2],A

LIST
LIST      #filespec
LIST      lno1[,lno2]
LIST      #filespec,lno1[,lno2]

LOMEM     adr

NEW

NUM
NUM       slno,incr
NUM       incr

PRINT
PRINT     #filespec
PRINT     lno1[,lno2]
PRINT     #filespec,lno1[,lno2]

REN       slno,incr
REN       incr

REP       /old string/new string/
REP       /old string/new string/,{A}
                                  {Q}
REP       /old string/new string/lno1[,lno2]
REP       /old string/new string/lno1[,lno2],{A}
                                             {Q}

SIZE
```

DEBUG

```
A       [adr]< assembler code     (blank required after <)

C       [adr1]< data

CR      <data

D
D       adr1[,adr2]

DR

G       [adr]

L
L       adr1[,adr2]

M       tadr < fsadr, feasr

S       [adr]

T       [adr]

V       adr1 < adr2,adr3

X
```

ASSEMBLER DIRECTIVES

```
.BYTE           expression and/or "string" list

.DBYTE          expression list

.END

.IF             expression,label

.INCLUDE        #filespec

.OPT            option list

.PAGE           ["string"]

.TAB            expression, expression, expression

.TITLE          "string"

.WORD           expression list

*=              expression

=               expression
```

ERROR SUMMARY

This is a summary of error messages produced by the EASMD program. For a more detailed decripition see the section on ERROR DESCRIPTION.

EASMD ERRORS:

1 - MEMORY FULL
2 - INVALID DELETE RANGE
3 - DEBUG ASSEMBLER ADDRESS ERROR
4 - BLANK REQUIRED AFTER LINE NUMBER
5 - UNDEFINED REFERENCE
6 - ASSEMBLER SYNTAX ERROR
7 - DUPLICATE LABEL
8 - BUFFER OVERFLOW
9 - EQUATE HAS NO LABEL
10 - VALUE OF EXPRESSION > 255
11 - NULL STRING
12 - INVALID ADDRESS OR ADDRESS TYPE
13 - PHASE ERROR
14 - UNDEFINED/FORWARD REFERENCE FOR *= (ORG)
15 - LINE TOO LONG
16 - INVALID INPUT LINE
17 - LINE NUMBER TOO BIG
19 - NO ORIGIN (*=) SPECIFIED
20 - OVERFLOW ON NUM OR REN
21 - NESTED INCLUDE INVALID

For the user convenience a summary of the error messages that can be generated by DOS and passed to EASMD are included.

DOS ERRORS:

DEC	HEX	MESSAGE
128	(80)	BREAK ABORT
129	(81)	FILE ALREADY OPEN
130	(82)	NON EXISTENT DEVICE
131	(83)	FILE OPENED FOR WRITE ONLY
132	(84)	INVALID COMMAND
133	(85)	DEVICE OR FILE NOT OPEN
134	(86)	INVALID IOCB NUMBER
135	(87)	FILE OPENED FOR READ ONLY
136	(88)	END OF FILE
138	(8A)	DEVICE TIMEOUT
139	(8B)	DEVICE NAK
144	(90)	DEVICE DONE ERROR
146	(92)	FUNCTION NOT IMPLEMENTED
160	(A0)	DRIVE # ERROR
161	(A1)	TOO MANY OPEN FILES (NO SECTOR BUFFER AVAILABLE)
162	(A2)	MEDIUM FULL (NO FREE SECTORS)
163	(A3)	FATAL SYSTEM DATA I/O ERROR
164	(A4)	FILE # MISMATCH
165	(A5)	FILE NAME ERROR
166	(A6)	POINT DATA LENGTH ERROR
167	(A7)	FILE PROTECTED
168	(A8)	COMMAND INVALID (SPECIAL OPERATION CODE)
169	(A9)	DIRECTORY FULL
170	(AA)	FILE NOT FOUND
171	(AB)	POINT INVALID