ACORN COMPUTER Acorn Computers Limited, 4a Market Hill, Cambridge CB2 3NJ, England. Telephone 0223 312772

ACORN TECHNICAL MANUAL.

Acorn 6502 ADE

Issue 1 Sept 1980.

Introduction

The Acorn ADE package is designed to run on 6502 based Acorns with at least 4K of memory. There are three major actions carried out by the package:

A TEXT EDITOR which is used to create source programs

AN ASSEMBLER which creates executable machine code from a symbolic source

A DISASSEMBLER which produces mnemonics and symbols from machine code

The program occupies memory between addresses #C000 and #D000 and is configured for use with 16K of RAM from #2000 to #4000 and an Acorn Operating System. If you have ADE in EPROM you should enter it by typing

```
GO C000
```

If you have ADE on cassette, you can enter

```
RUN "ADE"
```

to your COS to get it to load ADE and enter at #C000. If you have ADE on disk, you can enter one of

```
RUN "ADE"
RUN ADE
ADE
```

to get ADE loaded and enter it at #C000. If you wish to save ADE on cassette or disk, a suitable command is

```
SAVE "ADE" C000 D000
```

for either COS or DOS. ADE uses addresses in page 0 up to #0060 and all of page 1 for working values, it's addresses for text, symbol table and code destination may be changed by the user.

System Entry

Initial entry into the system will set default values for the positions of the text area, symbol table and object area and then execute the Clear command. The system will prompt with

```
New?
```

and, if you respond with Y or anything starting with Y, the text area will be formatted and the symbol table end will be set to it's beginning so that there are no symbols there. The editor prompt > then indicates that you are in the Editor command mode.

The Editor

The Editor is also the place where you can use the disassembler and assembler. Commands are specified by the first letter after the prompt >. Further letters or spaces are ignored until the argument(s) are encountered: these take the form of a list of numbers (possibly in hexadecimal) seperated by commas. If the argument is a hexadecimal number, then it may start with one of the characters A to F, in this case it is necessary for a space to seperate the command from the argument. Leading zeroes in hexadecimal or line numbers do not need to be specified. When entering any text into the system, there are two immediate editing functions; the first is the DELETE key which functions just as it does in the operating system, the second is CTRL-E which will copy out the previous text from the buffer. As an example, if you have typed a command like

```
*DELETE JIM
```

which you wish to repeat, you may type CTRL-E and the text will be written out again. If you have a teletext VDU, then the text given by CTRL-E will be in Magenta.

Editor Commands

A Add text to buffer/ Add text after line

The A command is used to add new lines of text to the end of the current source file, or to add up to nine lines of text after a particular line. If you type just A, then the system will prompt you with the highest line number and accept additional text lines until you type a line starting with CTRL-D. For example:

```
>A
0000: This is the first line of the document
0010: This is the next line
0020: and this is the last for now
0030:

>
```

If you type A and a line number, then the system will prompt you with the line number plus one, and accept up to nine lines (when it will make an Error 8D and you should renumber) or until you type a line starting with CTRL-D.

```
>A 10
0011: just after line 10
0012: another line after line 10
```

```
0013:

>
```

B Add text Before line

The B command is used to add up to nine lines of text before a particular line. If you type B and a line number, then the system will prompt you with the line number minus nine, and accept up to nine lnes (when it will make an Error 8D and you should renumber) or until you type a line starting with CTRL-D.

```
>B 30
0021: a line before line 30
0022:

>
```

C Clear text area

The C command may be used at any time to delete all the text. If you type C, then the system will type

```
New?
```

and only if you type Y or YES will your text be cleared. This prevents accidental deletion of text.

D Delete line or lines

The D command is used to delete a text line or a block of text lines. For example

```
>D 30
```

will delete line 30, and

```
>D 30,40
```

will delete all lines between lines 30 and 40.

E Edit line command

The E command is used to modify a single line, with the option of deleting following lines and of adding more lines after it. Typing E 30,40 is quivalent to typing D 30,39 and then E 40. If you type

```
>E 30
```

then the system will type out line 30 and prompt you with it's line number. You can either type in new information, or use CTRL-E to retreive the old information. Example

```
>C

New? Y

Clear
>A
0000: this is lien 1
0010: this ia line 2
0020: this is line 2
```

```
0030:

>N

>L
0010: this is lien 1
0020: this ia line 2
0030: this is line 2
0040:

>E 10
0010: this is lien 1
0010: NOW it IS line 1
0011:

>E 20,30

0020: this ia line 2
0030: this is line 2
0020:  this is line 2              typed CTRL-E
0021:

>L
0010: NOW it IS line 1
0020: this is line 2
0040:

>
```

If there is nothing you need to change on a line, type CTRL-E and RETURN. CTRL-E is useful for changeing the end of lines e.g. to change the third charater from the end type CTRL-E, DEL, DEL, DEL, new character, CTRL-E, RETURN. To change the first few characters type the characters and then CTRL-E to keep the rest of the line. If you have anything more complicated to do you can also use the operating systems cursor editor.

F Find string

The F command will find strings in the text file, either starting at the start or from a specified line. The string which you wish to find must be enclosed in two matched delimiters which do not occur in the string itself e.g.

```
>F/LDAIM /
```

or

```
>F;/;
```

Using this delimiter technique any string can be represented. The first delimiter must immediately follow the F. After the second delimiter you may give a line number from which the search will start, otherwise the search will start from the beginning of the text. If the string is found then the entire line containing the string is output, if the string is not found then the system will return to the command mode. Examples

```
>F/LDAIM /
0200:  LDAIM $FF
```

```
>F/abracadabra /

>
```

If you have just used an F command and it has found the line, then you may edit the line by just typing E (followed by a RETURN, of course).

G List preceeding 21 lines

The G command will list out the preceeding 21 lines from where you are currently in the text. It is complementary to the O command. If the current line is less than 200, then G will cause the first 21 lines (if there are 21) to be listed.

H Set address to data

The H command allows the user to set memory locations to values. H is followed by data and then address, both in hexadecimal. Only the least significant byte of the data is used. Example

```
>H 44,2E00
```

I Insert line(s) before line

The I command may be used to shift blocks of text around in the text file. You may either specify just one line to be moved, or an entire block of lines. Lines which have been moved are given line number zero, so it is important to renumber the lines after using I. Examples

```
>L
0010: ten
0020: twenty
0030: thirty
0040: forty
0050:

>I 10,40            insert line 40 before line 10

>L
0000: forty
0010: ten
0020: twenty
0030: thirty
0050:

>N

>L
0010: forty
0020: ten
0030: twenty
0040: thirty
0050:

>I 10,30,40         insert lines 30 to 40 before line 10

>N

>L
0010: twenty
0020: thirty
```

```
0030: forty
0040: ten
0050:

>
```

J Jump to pass two of assembler

The J command allows you to directly enter pass two of the assembler in order that you may use pass two to print a listing or generate object files, using the current contents of the symbol table. See the Assembler section for examples.

K Read in object file(s)

The K command acts like the R command, but issues a *USEX before each file access so that the files that it reads will be object files. Examples

```
>*MON

>K 1
X  UADE01  8000 C039  0025B 004

>K 2,4
X  UADE02  825B C039  0058C 01C
X  UADE03  87E7 C039  00607 022
X  UADE04  8DEE C039  00318 007

>
```

Since the assembler has saved each file with the correct load address, calculated from the ORG of the program, the files read using K will be completely executable.

L List

The L command will list the text file with line numbers. There are five options

```
L       will list all lines
L 10    will list line 10 only
L 10,40 will list lines 10 to 40 inclusive
L 40,   will list from line 40 to the end of the file
L ,40   will list lines up to line 40
```

A listing can be stopped by typing escape. Examples of the use of L can be found throughout this manual.

M Move memory around

The M command can be used to move 256 bytes or less of data from one location to another. The command

```
>M 2F00,2E00
```

will move the 256 bytes from 2F00 to 2FFF over to the bytes 2E00 to 2EFF. The command

```
>M 2F00,2E00,4
```

will move the 4 bytes from 2F00 to 2F03 over to the bytes 2E00 to

2E04. Because of the manner in which the M command works it is not possible to move a block to a lower address than its initial position if the end of the new block will overlap the start of the new block. If you wish to perform this type of move, then move the original data to a free page (e.g. page 0300) and then move from that page.

N Number all lines in increments of 10

The N command will renumber all lines in the text to a consecutive sequence of tens, starting at line number 0010. The N command should be used after the A, B, D, E, I commands to tidy up the text.

O List next 21 lines

The O command lists the next 21 lines from the current position in the text. It is useful to use O after an F command to show the environment, or to use O to step through the file slowly. A simple way to set the current line to n is to try and list line (n-1).

P Print lines

The P command functions in exactly the same way as the L command, except that the text is output without line numbers. The P command allows you to use the editor system for writing purely textual documents.

Q Contents of location

The Q command will output the byte in hexadecimal of the specified locataion. Example

```
>Q 2E00
A9
>
```

R Read source file

The R command will read in one or many (for verification purposes) source files. A source file has an 'ID' of two hex digits, excluding 00.

```
>*MON

>R 1
   UADE01   3000 C039  024E1 0BA

>R 2,4
   UADE02   3000 C039  02543 109
   UADE03   3000 C039  02CCB 077
   UADE04   3000 C039  01617 029

>
```

S Save source file

The S command will save the current text as a source file, or it can be used to save any specified block of memory as a source file. The 'ID' of the source file is specified in the S command and consists of two hex digits, excluding 00. Examples

```
>*MON

>L
```

```
0010 a test file
0020:

>S 20
3000 3012    UADE20  3000 C039  00012 01A

>S 21,400,500
0400 0500    UADE21  0400 C039  00100 01B

>
```

The system writes out the start and end addresses of the file it is saving. Files should be deleted using *DELETE.

T Type symbol table

The T command deals with the symbol table created by the assembler. There are various options

```
T 0,x    type symbol table alphabetically ordered in x+1 columns
T 1,x    type symbol table numerically ordered in x+1 columns
T 2      type symbol table start and end addresses
T 3,xxxx set symbol table end to xxxx
```

Useful values for x are 2 (for 40 characters wide) and 5 (for 80 characters wide). The ability to discover and set the symbol table end address allows a symbol table to be saved and reloaded. Each symbol defined by the source uses six bytes of space in the symbol table, so that 170 symbols can be used even if the symbol table space is restricted to 1K.

U Upper address used

The U command will type out the current end address of the text, together with the highest line number.

```
>U
3012 0020
>
```

V View 256 bytes of memory

The V command will type out 256 bytes of memory putting a specified number of bytes on the line. For example

```
>V 2E00,8
```

will type out 32 lines each with 8 bytes of data. A useful value for the number of bytes on the line is D (i.e. 13).

W Where is line

The W command returns the starting address of the line specified. You may then use the H, Q, V commands to insert or remove characters which are either impossible to type, or (like Form Feed) prevent simple use of the E command.

```
>W 20
3022 0020: BB ORG $8000

>
```

X Xecute

The X command can be followed by an address, in which case the system calls a subroutine at that address, or by nothing, in which case the assembler is entered at pass one.

Z Disassemble

The Z command enters the symbolic disassembler. The disassembler is useful for debugging or modifying programs for which there is no source code, or for inspecting what the assembler has actually done. There are three modes of use

```
Z       disassemble from where you left off
Z x     disassemble from address x
Z x,y   disassemble from address x to address y
```

The last mode outputs the entire disassembly in one go (though it can be stopped by ESC) while the first two output 25 lines and then wait for any key. If the key is a RETURN, they return to command mode, else they continue. The disassembler uses the symbol table which has been built by the assembler to give symbolic names to all those arguments that it recognizes. If you do not have the source of the program, then you can write a source file consisting of define symbol operations and assemble it to provide a symbol table. The disassembler can make things highly visible, to find all references to OSWRCH, simply define only that symbol and use the disassembler.

* Call operating system

The * command will take the string on its right and give it to the operating system as a command in the normal way.

Error handling

The errors in the ADE system are signalled using BRK, the error number is just the low byte of the PC on stack. Errors found by the operating system are described and then a BRK is caused to return. Users can set BRK instructions in code to cause a return to the editor so that variables (not, however, A, X and Y) can be inspected).

The Assembler

The assembler is designed to make programming the 6502 microprocessor as easy as possible. A source program is created using the text editor and following the format described below. If the program is short, it can reside in the text space, if it is long (greater than about 512 bytes of assembled code) it will probably exist as a set of seperate files. When executed, the assembler translates the source statements into the equivalent 6502 machine code instructions, using two distinct steps. During pass one, the first step, the entire source is read by the assembler in order to generate a symbol table consisting of the names of all the symbols used and their equivalent hexadecimal values. During pass two, the second step, the entire source is again read and this time machine executable object code is generated. All the time that you are 'in' the Assembler, leading zeroes in hexadecimal numbers are required.

Source format

The input data for the assembler is formatted in blocks of variable length records. Each record contains a line number, a variable amount of data and a terminating carriage return (0D). The entire block is terminated by a CTRL-D end of text character. Each source statement for the assembler is divided into five fields, the label, the instruction, the address mode, the argument and the comment. Each field is delimted by a single space excepting the instruction and address mode boundary. In many cases a field may not be present in which case its absence must be shown by leaving a single space or terminating the statement with RETURN. It is important to remember that spaces are used as delimiters and that the number of spaces in a line (before getting to the comment) is critical.

Statement format

label		instruction	address mode		argument		comment

The label field

Any statement may be identified with a symbolic label containing up to six alphabetic characters. The label must always begin in the first column of the statement line. All labels must be uniquely defined and the assembler will flag duplicated labels. Unless the define symbol operation is used, the value of the symbol will be the current address as calculated by the assembler for the statement. Symbols referring to page zero must be defined before they are used so that the assembler can correctly calculate the number of bytes used by an instruction on its first pass. It is generally considered good programming practice to define all data symbols at the beginning of the program.

```
Valid symbol usage      Invalid symbol usage
DATA LDA FRED           DATA1 LDA FRED2
TEST = $03               TEST = 03
SUB * TEST +01          SUB +01 * TEST
```

The instruction field

The second field of each source statement is the instruction field which is immediately preceeded by a space. Instructions consist of three character mnemonics for 6502 instructions as in the 6502 programming manual (except that JMP indirect has a mnemonic JMI) or a pseudo instruction. A complete table of instructions and their valid address modes is given below.

	IM	A	AX	ZX	AY	ZY	IX	IY	Z	(abs)	(rel)	(imp)
ADC	X		X	X	X		X	X	X	X		
AND	X		X	X	X		X	X	X	X		
ASL		X	X	X					X	X		
BCC											X	
BCS											X	
BEQ											X	
BIT									X	X		
BMI											X	
BNE											X	
BPL											X	
BRK												X
BVC											X	
BVS											X	
CLC												X
CLD												X
CLI												X
CLV												X
CMP	X		X	X	X		X	X	X	X		
CPX	X								X	X		
CPY	X								X	X		
DEC			X	X					X	X		
DEX												X
DEY												X
EOR	X		X	X	X		X	X	X	X		
INC			X	X					X	X		
INX												X
INY												X
JMI										X		
JMP										X		
JSR										X		
LDA	X		X	X	X		X	X	X	X		
LDX	X				X	X			X	X		
LDY	X		X	X					X	X		
LSR		X	X	X					X	X		
NOP												X
ORA	X		X	X	X		X	X	X	X		
PHA												X
PHP												X
PLA												X
PLP												X
ROL		X	X	X					X	X		
ROR		X	X	X					X	X		
RTI												X
RTS												X
SBC	X		X	X	X		X	X	X	X		
SEC												X
SED												X
SEI												X
STA			X	X	X		X	X	X	X		
STX						X			X	X		
STY				X					X	X		
TAX												X
TAY												X
TSX												X
TXA												X
TXS												X
TYA												X

The pseudo instructions

There are three pseudo instructions which may be used and which generate no machine code. The first is ORG which defines the origin address of the program section. There should be an ORG at the beginning of the source, and at most one ORG per file. A label on the ORG statement will be part of the header line of a printed listing.

The second pseudo instruction is * which defines the label field as equivalent to the following argument field. Define symbol operations are the only type of operation which should precede the first ORG of the source file(s).

The last pseudo instruction is = which directly defines a single byte in the object code as equal to its argument.

```
TEST ORG $2E00
ZERO * $0000
THREE * ZERO +03
QMARK * '?
INITC = '!    initial character
```

The address mode

The address mode consists of zero, one or two characters immediately following the instruction field. In the absolute, relative, or implied address modes the assembler is able to work out the required mode from the instruction field and so no address mode is required. The valid address modes are:

A Accumulator addressing. The instruction operates on the accumulator.

IM Immediate. The operand of the instruction is the argument following. The argument may be anything.

AX Absolute indexed by X. The operand of the instruction is the address represented by the argument added to the value of the X index register. If the argument is a page zero location and a valid page zero instruction exists, then the assembler will substitute the ZX mode.

ZX Zero page indexed by X. The operand of the instruction is the address represented by the argument added to the value of the X index register. The high bytes of the address is ignored and the effective address will always be in page zero.

AY Absolute indexed by Y. The operand of the instruction is the address represented by the argument added to the value of the Y index register. If the argument is a page zero location and a valid page zero instruction exists, then the assembler will substitute the ZY mode.

ZY Zero page indexed by Y. The operand of the instruction is the address represented by the argument added to the value of the Y index register. The high bytes of the address is ignored and the effective address will always be in page zero.

IX Indexed indirect. The argument address is added to the X index register forming an address in page zero. The contents of this location and the next one form the address of the operand for the instruction.

IY Indirect indexed. The argument addresses a location in page zero. The sum of the contents of the address in this location and the next together with the Y index register forms the address of the operand for the instruction.

Absolute. The effective address is given directly by the argument, if this is in page zero and a valid page zero instruction exists, then the assembler will automatically substitute the Z mode.

Z Zero page. The argument is an address in zero page, its high byte is ignored.

Relative. Relative instructions can branch to within 128 bytes of the current address. The assembler calculates the required distance from the value of the argument.

Implied. Implied addressing requires no specification since all operands are specified by the instruction.

Indirect. There is no indirect mode in the assembler since the JMP indirect instruction has been replaced by the JMI instruction with absolute address mode.

The assembler will detect most common address mode errors, although totally illogical combinations (e.g. ASLIM) may defeat it.

The argument field

The argument field is used to define the operand for an instruction or pseudo instruction. There are three types of argument, symbolic (labels), hexadecimal, or character.

Symbolic arguments

Symbolic arguments are symbols defined elsewhere in the source. The equivalent address or data is substituted for the symbol in the object code. In order to reduce the size of symbol tables a symbolic argument can consist of a symbol with a modification operation after it. There are three types of modification operation: adding and subtracting a two digit hexadecimal constant, a special 'take other byte' operator, / and a special 'set top bit' operator, !. For example

```
Source statemnt      Assembly listing
FRED * $1234           0200             FRED   *      $1234
 LDAIM FRED            0200 A9 34              LDAIM  FRED
 LDAIM FRED /          0202 A9 12              LDAIM  FRED   /
 = FRED                0204 12                 =      FRED
 = FRED /              0205 34                 =      FRED   /
 STA FRED              0206 8D 34 12           STA    FRED
 STA FRED -01          0209 8D 33 12           STA    FRED   -01
 STA FRED +01          020C 8D 35 12           STA    FRED   +01
 = FRED !              020F 92                 =      FRED   !
```

Note that the value used for + and - must always be two digits long.

Hexadecimal arguments

Hexadecimal arguments are identified by a dollar sign as the first character of the argument field. Either two or four hexadecimal digits must follow the dollar. If only two digits are used, then the value will be duplicated in both bytes of a symbol value.

Character arguments

The value in ASCII of any character may be found by preceeding it with a ' mark. The value will be duplicated in both bytes of a symbol value.

The comment field

The last field of a source statement is simply a comment which will be printed out in assembly listings. To put a comment on a line by itself, precede it by three spaces.

Operating the assembler

Enter the assembler from the editor command mode using X

```
>X

Pass 1
ID
```

If the source file is in memory, enter 00; if the source is a sinlge file, enter its ID, noting that no leading zero suppression is allowed at this stage. The assembler will carry out a pass one on the source, writing out the current ID as it reads each new source file. Any detected errors will be output, together with the source line where the error was found. The 'ID ' prompt will reappear when this task is complete. You may continue doing pass ones on further source files by typing IDs in response to this prompt, or you may type RETURN

```
>X

Pass 1
ID 01,03

ID 02
ID 03
ID

Pass 2
Print?
```

If you wish to have an assembly listing printed type something starting in Y (for example Y followed by CTRL-B to start the printer). Anything else is taken as no.

```
>X

Pass 1
ID 01,03

ID 02
ID 03
ID

Pass 2
Print? Y

Save ID
```

If you wish to save the object code generated to be saved as files, then enter a valid ID (01 to FF). After each file's code is generated,

the assembler will generate a USEX command to the OS and save the code generated with the start address set for execution of the code. Multiple input files will generate multiple object code files, the Save ID being incremented after every object code save. In the COS the USEX command is ignored so the Save ID should be carefully selected, or code collected on a special record only drive. In the DOS it is useful to specify the same Save ID as the source file ID to make the source to object correspondance clear. The assembler will now output the 'ID ' prompt again, and it should be given exactly the same responses as at pass one. Detected errors will be printed out regardless of whether a listing is required or not. At the end of pass two, you will be returned to the editor command mode. The J command enters pass two at its start and can be used to get a listing of the program which was just cesfully assembled. The assembly listing is neatly formatted and divided up into pages each of which is preceded by a Form Feed and carries a header line at the top.

Object format

The object code generated by the assembler is stored in the object code area. For multiple source files the code is repetetively overwritten for each file. However, if you have assembled just one file, then the entire code will be resident in the object code area and can be executed (if its ORG is correct) or Moved to the correct address. Object code files saved by the assembler have the correct reload address to simply be loaded back into memory for execution. Generally this can be done using the K command, but if the program is to run in the area occupied by ADE, then ADE must be left (e.g. press BREAK) and the files read in using the operating system. Use of Save ID FF can cause problems when later using the K command.

Memory Allocation

The positions and sizes of the text area and symbol table and the position of the object code area are defined in a table in ADE, held at CF62. The six bytes in this table are copied into zero page addresses 0 to 5 on initial entry to ADE and they are subsequently only referred to. Users with ADE in EPROM wishing to change the memory allocation will need to change these bytes each time they enter ADE. Users with ADE on disk or cassette can change the bytes at CF62 and save the new version. The bytes are

```
SOURCM = $2F    text start minus one
SOURCE = $30    text start page
SOURCF = $60    text end page plus one
SYMBOL = $22    first page of symbol table
SYMF   = $2E    last page of symbol table plus one
OBJECT = $2E    first page of object code area
```

as supplied. This allocation uses 16K of RAM for the system. Users with more RAM available are advised to reallocate the symbol table since it is only two and a half Kbytes and is situated where the disk system keeps sequential file buffers. Users with only 8K of RAM should make SOURCF $40. Although this allocation allows the object code to overwrite the source, since there is a compression of at least a factor of 8 when a file is assembled, the object code will never catch up with the source.

Example of system use

```
New? Y

Clear
>A
0000:     Message program
0010:     Call by JSR followed by string of bytes
0020:     terminated by a negative code which is
0030:     then executed.
0040: EXAMPL ORG $2E00
0050: PTR * $0080
0060: OSWRCH * $FFF4
0070: STRING PLA
0080:  STA PTR
0090:  PLA
0100:  STA PTR +01
0110:  LDYIM $00
0120: STRLOP INC PTR
0130:  BNE NOINC
0140:  INC PTR +01
0150: NOINC LDAIY PTR
0160:  BMI STREND
0170:  JSR OSWRCH
0180:  JMP STRLOP
0190: STREND JMI PTR
0200:
0210: TESTER JSR STRANG
0220:  = 'H
0230: = 'I
0240:  = $0A
0250:  = $0D
0260:  NOP
0270:  RTS
0280:

>X

Pass 1
ID 00

 Error D5 0230 =        'I
ID                           |pressed ESC
>E 230
0230: = 'I
0230:  = 'I
0231:

>X

Pass 1
ID 00

ID

Print?

Save ID

ID 00
```

```
 Error 6E 0210
0210: 2E?? 00 00    TESTER JSR   STRANG
ID

>E 210
0210: TESTER JSR STRANG
0210: TESTER JSR STRING
0211:

>X

Pass 1
ID 00

ID

Pass 2
Print?

Save ID

ID 00

ID

>J

Pass 2
Print? Y

Save ID

ID 00
EXAMPL   ACORN 6502 Assembler  Page 01


0000:                Message program
0010:                Call by JSR followed by string of bytes
0020:                terminated by a negative code which is
0030:                then executed.
0040: 2E00           EXAMPL ORG   $2E00
0050: 2E00           PTR    *     $0080
0060: 2E00           OSWRCH *     $FFF4
0070: 2E00 68        STRING PLA
0080: 2E01 85 80            STA   PTR
0090: 2E03 68               PLA
0100: 2E04 85 81            STA   PTR     +01
0110: 2E06 A0 00            LDYIM $00
0120: 2E08 E6 80     STRLOP INC   PTR
0130: 2E0A D0 02            BNE   NOINC
0140: 2E0C E6 81            INC   PTR     +01
0150: 2E0E B1 80     NOINC  LDAIY PTR
0160: 2E10 30 06            BMI   STREND
0170: 2E12 20 F4 FF         JSR   OSWRCH
0180: 2E15 4C 08 2E         JMP   STRLOP
0190: 2E18 6C 80 00 STREND JMI    PTR
0200:
0210: 2E1B 20 00 2E TESTER JSR    STRING
0220: 2E1E 48               =     'H
```

```
0230: 2E1F 49              =     'I
0240: 2E20 0A              =     $0A
0250: 2E21 0D              =     $0D
0260: 2E22 EA              NOP
0270: 2E23 60              RTS

ID

>X 2E1B
HI

>Z 2E00,2E24

2E00 68           EXAMPL PLA
2E01 85 80                STAZ  PTR
2E03 68                   PLA
2E04 85 81                STAZ  $81
2E06 A0 00                LDYIM $00
2E08 E6 80        STRLOP  INCZ  PTR
2E0A D0 02                BNE   NOINC
2E0C E6 81                INCZ  $81
2E0E B1 80        NOINC   LDAIY PTR
2E10 30 06                BMI   STREND
2E12 20 F4 FF             JSR   OSWRCH
2E15 4C 08 2E             JMP   STRLOP
2E18 6C 80 00     STREND  JMI   PTR
2E1B 20 00 2E     TESTER  JSR   EXAMPL
2E1E 48                   PHA
2E1F 49 0A                EORIM $0A
2E21 OD EA 60             ORA   $60EA
>T0,2

Symbol Table 2E00 2E30
EXAMPL 2E00 NOINC  2E0E OSWRCH FFF4
PTR    0080 STREND 2E18 STRING 2E00
STRLOP 2E08 TESTER 2E1B
>T1,2

Symbol Table 2E00 2E30
PTR    0080 EXAMPL 2E00 STRING 2E00
STRLOP 2E08 NOINC  2E0E STREND 2E18
TESTER 2E1B OSWRCH FFF4
>
```

Brief List of Editor Commands

```
A         Add text at end of text buffer
A XX      Add text after line XX
B XX      Add text before line XX
C         Clear text area
D XX      Delete line XX
D XX,YY   Delete lines XX to YY
E XX      Edit line XX
F/jim/    Find jim starting at line 0  | the / around the string
F/jim/XX  Find jim starting at line XX | may be any character
G         Go back 21 lines and list out 20 (by line number)
H AA,BBB  Hex AA to location BBB (one byte only)
I XX,YY   Insert line YY before line XX     | need to renumber
I X,Y,Z   Insert lines Y to Z before line X | after this
J         Jump to Pass 2 of the Assembler
K XX      Read in assembler object file XX (disk name X  UADEXX)
K XX,YY   Read in assembler object files XX to YY
L         List text
L XX      List text line XX
L XX,YY   List text lines XX to YY
L ,YY     List text lines from start to YY
L XX,     List text lines from XX to end
M AA,BB   Move 256 bytes from AA to BB
M A,B,C   Move C (C less than 256) bytes from A to B
N         Number text lines in increments of 10
O         List onward 21 lines
P         Print text without line numbers
P XX      Print text line XX
P XX,YY   Print text lines XX to YY
P ,YY     Print text lines from start to YY
P XX,     Print text lines from XX to end
Q AA      Write contents of AA
R XX      Read source file UADEXX
R XX,YY   Read source files UADEXX to UADEYY
S XX      Save current text as file UADEXX
S XX,A,B  Save memory A to B as file UADEXX
T 0,X     Type symbol table alphabetically ordered in X+1 columns
T 1,X     Type symbol table numericaly ordered in X+1 columns
T 2       Type symbol table start and end addresses
T 3,XX    Set symbol table end to XX
U         Type upper address of text
V AA,BB   Type 256 bytes of memory from AA putting BB bytes on a line
W XX      Type where in memory line XX is held
X         Xecute the assembler
X AA      Xecute machine code from address AA
Y         Outputs Error XX where XX is the starting page of the program
Z         Disassemble from where you left off | operates in 25
Z AA      Disassemble from AA                 | lines per page
Z AA,BB   Disassemble from AA to BB
*....     Give string .... to operating system
```

During buffer entry:

```
ENQ (ctrl-e)  echoes last buffer
DEL (delete)  deletes last character
CAN (ctrl-x)  deletes buffer
```

Assembler Errors

3 Insufficient memory to insert line

1D Illegal line number in I command

45 Attempt to redefine symbol

6E Unknown symbol

8D Insertion overflow

92 Command syntax

B2 Unknown addressing mode

BF Symbol table overflow

C0 Y command

C9 Branch out of range

D5 Unknown instruction type

F1 Impossible address mode

F7 Unknown command