
VIZA
SOFTWARE
Edwa
PROFESSIONAL ASSEMBLER

Pagina Bianca

Ready64.org (2024)

Copyright Notice

Disclaimer

VIZA SOFTWARE LIMITED takes no responsibility for any errors or omissions in the EDNA software program or associated manuals. While every effort has been taken to ensure that the product is suitable for its intended purpose, VIZA SOFTWARE LIMITED can take no responsibility for any loss or damage incurred through use of this product, and disclaims any implied warranties of suitability or fitness for any purpose.

Conditions Of Use

The EDNA license grants the licensee permission to install and operate the EDNA software on a single computer owned or used by the licensee. **The EDNA software product shall not be lent, sold, exchanged or copied to any third party under this licensing agreement.**

Credits

EDNA was researched, designed, programmed and documented by Anthony Robinson. This manual was written and compiled by Anthony Robinson, Jon Dunn and Kelvin Lacy.

Pagina Bianca

Ready64.org (2024)

Contents

1. Getting Started

2. The Editor

3. The Assembler

4. The Monitor

5. Using Machine Code Within A BASIC Program

Appendix

Index

Pagina Bianca

Ready64.org (2024)

Chapter 1.

Getting Started

Welcome

Welcome to EDNA, the first integrated editor, assembler and monitor for the Commodore 64 and 128. EDNA has been designed to allow the simultaneous development of both machine code AND BASIC. EDNA is always at hand, the editor is used to enter the machine code program lines, the assembler collates and translates these into machine instructions, and the monitor assists in getting the bugs out. EDNA can't write your programs for you, but it can do practically everything else !!

This reference manual details the commands and use of the EDITOR, ASSEMBLER and MONITOR. We have included definitions of terms and explanations of 6502 addressing modes, and the supplied disk contains sample source files and a conversion program that translates Commodore and Mikro source files.

This is a reference manual, but can also be read from cover to cover. EDNA is very easy to use, the command and control keys are logically allocated, the functions of EDNA are simple, yet comprehensive. This manual is not intended to cover the 6502 programming language or Commodore operating system. There are many publications that cover these areas. For beginners we have included examples on the supplied disk. Use these to get some confidence and experience. EDNA has been used to develop and maintain VIZAWRITE, VIZASTAR and, of course, EDNA itself !!

Writing machine code is easy, getting it to work is the hard part !!

How To Start

First switch off the computer, disk drive and printer. Insert the cartridge (label upwards) into the cartridge slot. Turn on the printer, turn on the disk drive wait a few seconds, finally turn on the computer. The Commodore 64 will 'power-up' in the machine code monitor.

Getting To The Editor

To switch to the Editor from the Monitor press RUN. This clears all the available memory, to give an empty source file. Subsequent entries to the Editor show the last source file used. To return to the Monitor from the Editor press RUN once again.

Getting To BASIC From The Monitor

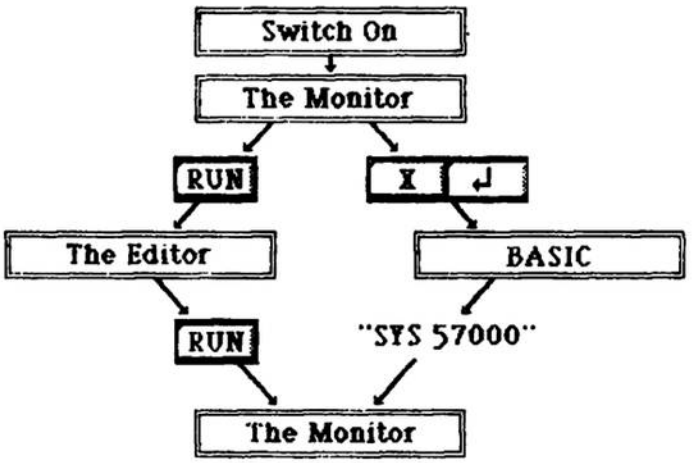

Figure 1.1 Route Map

Two different methods of using BASIC are provided:

1. PROTECT SOURCE - X (RETURN)

This mode of operation allows both BASIC and EDNA source files to co-exist in memory. You can then swap between them at will. The amount of BASIC memory is reduced to 20223 bytes. The EDNA source file and BASIC program, plus variables, are preserved when switching. Return to the monitor with SYS 57000 from BASIC.

2. DON'T PROTECT SOURCE - SHIFT X (RETURN)

The full 38911 bytes of BASIC memory are available in this mode. BASIC program source and EDNA source files will occupy the same memory and can therefore corrupt each other. Return to the Monitor with SYS 57000 from BASIC.

Chapter 2.

The Editor

Getting To The Editor

EDNA 'powers-up' in its machine code monitor, from where you may enter the Editor or BASIC. To select the Editor press RUN *. Obviously, at switch-on there is no source file in memory. However, subsequent entries to the Editor returns back to the current source file in use. Switch to the Monitor by pressing RUN from within the Editor.

When you first enter the Editor you are required to select the storage device required. The current device will be highlighted in reverse characters. To change the device: press a 'D' for disk, or a 'C' for cassette. Press RETURN to continue.

You will be then be prompted to select the type of printer in use. The options available are:

P - Parallel (Centronics) printers connected via the User Port.
S - Commodore printers connected via the serial port.
L - Line Feed (after Carriage Return).

Select the printer type and line feed option and press RETURN. Press 'L' to turn on the Line Feed option, press it again to turn line feeds off.

The line feed option sends a LINE FEED to the printer after each CARRIAGE RETURN. Commodore printers generate their own line feeds; other printers can usually be switched for either option. Details of parallel printer connections are given in the Appendix.

Using a Parallel Printer Within A BASIC Program

EDNA allows a parallel printer to be used within BASIC. POKE 154,128 (192 if you need line feeds) switches all output to the printer. POKE 154,3 restores normal output to the screen.

*To enter BASIC type 'x' and RETURN (or SHIFT 'x'). Entry to BASIC is further explained in Chapter 1 - Getting Started - page 1-2.

The Memory Scale

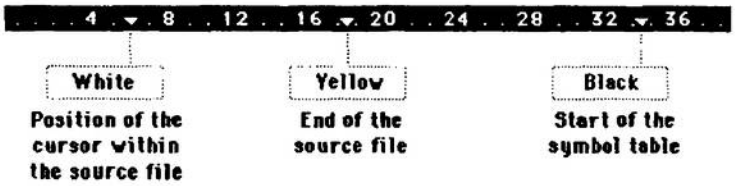

Figure 2.1 The Memory Scale

The scale on the second row of the screen shows how much memory is available for source and symbol table. The graduations are in Kilobytes up to a maximum of 38K of memory. The source file is stored from the bottom of the file memory working up: the symbol table is stored from the top of the file memory working down. On the scale are three triangular pointers which represent:

White	the position of the cursor within the source file
Yellow	the end of the source file
Black	the start of the symbol table (after assembly)

The separation of the Yellow and Black pointers on the scale shows the amount of spare memory available. Initially, with no file in memory, the yellow pointer will not be visible, the white pointer will be on the far left of the scale and the black pointer on the far right.

As a rough guide, the source file will usually be about four to six times the size of the object file, depending upon the number of comments used and the length of labels. The symbol table will usually be about half the size of the object file. So, it possible to assemble files of approximately 8K from a single source file in memory. Larger programs will need to use one of the disk file linking options (see the Assemble Command description in Chapter 3).

FIND, GOTO, REPLACE, LIST and ASSEMBLE commands show their progress through the source file by moving the white triangle. Note that DISK and serial PRINTER operations turn the triangles off.

The Editor

Editing could not be simpler. There are no line numbers to contend with. Just type your source line on the screen. The cursor can be moved onto a character anywhere on the screen using the normal cursor control keys. Characters can be typed-over, deleted or inserted. The Editor commands allow whole ranges of lines to be copied, moved or deleted. The FIND and GOTO commands are used to instantly locate sections of source.

A line of text enters the source file as the cursor is moved to another line. The source file can be scrolled through, by moving the cursor past the top or bottom of the screen.

Lines may be up to 80 characters long, the line scrolls sideways as the cursor is moved to the edge of the screen.

As the line is entered into the source file checks are performed on the syntax of the line:

i) Labels must start in the left margin with an upper or lower case letter of the alphabet and be no more than six characters long.

ii) The op-code must be a recognised 6502 instruction or EDNA directive. For example:

```
LDA  #'E'
.BYT 0
```

iii) Any ASCII text in the operand must have both preceding and closing quotation marks. For example:

```
CMP  #'A'
.BYT 'apostrophies'
```

iv) A space between a label and an '='. For example:

```
START =$2000
```

If the syntax is incorrect a 'beep' is sounded and the cursor positioned over the error for correction.

Text Editing Keys

CRSR keys	Move the cursor in the indicated direction.
RUN	Exit to the machine code monitor.
STOP	Cancel the current line entry, restore any previous entry, or cancel the current command.
CLR	Erase everything to the right of the cursor on the current line.
HOME	Move the current line to the centre of the screen.
INST	Insert a space at the cursor position.
DEL	Delete the character to the left of the cursor.
RETURN	Place the cursor at the start of the next line, or tab to the op-code field.
SHIFT RETURN	Next Find/Replace.
CTRL	Pauses listing when held down.
F1	Go to the end of the file.
F2	Go to the start of the file.
F3	Next screen.
F4	Previous screen.
F5	Place the cursor at the start of the next field.
F6	Place the cursor at the end of the text on the current line.
F7	Inserts a blank line above the cursor. Holding this key down produces a block of blank lines, allowing entry of a number of source lines.
F8	Deletes a line or several lines starting from the current cursor position. Use the CRSR keys to highlight the lines to be deleted, the F3 and F4 keys will cause whole screenfulls of information to be highlighted. Press the RETURN key to delete the lines from memory or STOP to cancel the command.

Use the STOP key to abandon all active commands. Also use this key to cancel changes made to the current line of text before they are entered into the source file (changes are stored in memory when the cursor is moved off the line). The STOP key also cancels all FIND and REPLACE modes.

Editor Command Summary

Keystroke	Description	Page/ref.
CBM A	Assemble Source File	2-6
CBM C	Copy Source Lines	2-8
CBM D	Disk Commands	2-8
CBM F	Find a Sequence Of Characters	2-9
CBM G	Go to Label	2-10
CBM I	Set Default input/output device	2-10
CBM L	Load Source File	2-11
CBM M	Move Source Lines	2-11
CBM N	New file - clear file & symbol table memory	2-12
CBM O	Old file - restore file erased by CBM N	2-12
CBM P	List file to screen/printer	2-12
CBM Q	Query. Programmer's Calculator.	2-13
CBM R	Replace text	2-14
CBM S	Save file	2-14
CBM X	Mark Source File Position	2-15
CBM Y	Go to Marked Source File Position	2-15
CBM Z	Append Source File	2-16
CBM 1	Set foreground colour	2-16
CBM 2	Set background colour	2-16
CBM 3	Set border colour	2-17
CBM 4	Set default screen colours	2-17

Editor Commands

1. Assemble

Purpose: Assemble the current source file.

Command: CBM A

Sequence: Hold down the CBM key and press 'A'. Enable any required assembly option by pressing its first letter key, press the key again to disable the option. Press RETURN to start assembly. After assembly, press any key to clear the display and return to the source file editor. Press STOP to terminate assembly.

Notes: The options available are:

(default) - Errors Only.
This is the default assembly option. The assembler then lists only lines containing errors. A descriptive message accompanies each line listed. After assembly - pressing any key returns back to the source file display.

P - Printer.
Enable this option and **LIST** to print the assembled program. All 80 columns of each source line are printed, as opposed to just 40 columns shown on the screen display. EDNA assumes a 66-line page, and performs page skips automatically, unless instructed by a **.PAGE** directive.

F - Full Assembly Listing.
The Assembler directives **.BYT**, **.WOR** and **.DBY** often cause more object code to be generated than can be listed on one line. This option lists all the machine code generated by the directive. Normally, just the first line of machine code is listed. Lines containing errors are automatically listed in full.

L - List Assembled Source Lines

Two extra 'fields' of information are added to the source code listing on the left, detailing the memory address and **object code** of the current instruction. These are the hexadecimal values of the machine code program. These values take up 14 columns on the printer, making it necessary to restrict source lines to 66 columns where an 80 column printer is in use. If this is not done, the printer automatically carriage return's at the end of a long line, splitting it into two and causing EDNA's internal line count to be incorrect. This causes the listing to 'slip' down over the paper perforations.

S - Symbol Table.

This option lists all symbols and values at the end of an assembly. Symbols which have been defined but are not referenced will will be marked with an asterisk ('*'). This information can be used to draw attention to any redundant sections of code.

M - Machine Code To Memory.

This option places the machine code directly into memory as it is assembled starting from the address defined in the source file. **Take Care !** If this option is used carelessly, it will corrupt the source file or symbol table. The recommended area is the spare 4k block from $C000 to $CFFF. This area is not used for BASIC programs, so routines at this address may be used with BASIC SYS calls. The 8k areas of memory from $A000 to $BFFF (under the BASIC ROM) and from $E000 to $FFFF (under the Kernal ROM) may be used if BASIC files are not being PROTECTED (See X command - Chapter 1 - Getting Started - page 1-4).

D - Machine Code To Disk.
Use this option to write the assembled machine code directly out to a disk file. The disk filename is taken from the source filename, and is then prefixed by a full-stop. The file automatically replaces any other file of the same name on the disk. The **load** address of the file may be changed by using the '**•=**' directive (see Chapter 3 - Assembler Directives).

2. Copy

Purpose: To highlight a range of source lines in one part of the source file and copy them to another location anywhere in the same source file. Both parts of the source file will then contain the same source lines.

Command: CBM C

Sequence: Place the cursor on the first line of the range to be copied, hold down the CBM key and press 'C'. Highlight the lines to be copied and press RETURN. Move the cursor to the beginning of the new location and press RETURN.

Notes: Source lines are highlighted with the CRSR up, CRSR down, F3 and F4 keys. Hold them down to repeat.

3. Disk Command

Purpose: To allow access to the CBM Disk Operating System (DOS) commands. These commands are fully detailed in the CBM disk manual.

Command: CBM D

Sequence: Hold down the CBM key and press 'D', press RETURN to display the disk directory, or type a disk command in the correct format and press RETURN.

Notes: The default command is '$0', this displays the disk directory. Press the CTRL key to pause display of a long directory list. To exit the disk command mode press the STOP key.

To repeat a disk command press RETURN again.

The following are most likely to be of use:

$0: na*	Selective directory.
N0: name,id	Format and header whole disk
N0: name	Rename disk and clear directory
S0: name	Scratch file from directory
S0: na*	Scratch family of files, take care !!
V0	Validate disk
R0: newname=oldname	Rename file (not to be confused with N)

4. Find

Purpose: To search through the source file for a sequence of characters.

Command: CBM F

Sequence: Hold down the CBM key and press 'F'. Enter the sequence of characters to be found and press RETURN.

Notes: The search starts from the the current cursor location. When a match is found - the cursor marks the start of the found character sequence. The Editor may be used as normal from this point. The 'FIND:' prompt continues to be displayed at the top of the screen to allow repeated searches for the specified sequence. Press F2 function key to repeat the search from the start of the file.

To resume the search for the next occurrence: Press SHIFT and RETURN together. The triangle on the memory scale indicates the progress of the search through the file.

To change the search character sequence: issue the FIND command again.

The FIND command can be abandoned by entering one of the other search modes, GOTO or REPLACE, or by pressing the STOP key. The STOP key will first cancel the current line entry; if the current line has been changed **move the cursor off the line before abandoning the command**.

5. Goto

Purpose: To search the source file for a specific label, at the start of a source line.

Command: CBM G

Sequence: Hold down the CBM key and press 'G', enter the label name and press RETURN.

Notes: The search is made for the occurrence of a label in the 'label field' of a line (i.e. its definition), a match is **not** be made where the label occurs in the operand field, or elsewhere.

6. Input/Output Device

Purpose: To select the current Input/Output device to disk or cassette. To select serial or parallel printer output.

Command: CBM I

Sequence: Hold down the CBM key and press 'I', type 'C' (for cassette) or 'D' (for disk) and press RETURN. Type 'P' or 'S' to change to a parallel or serial printer. Type 'L' to switch the line feed option on or off. Press RETURN.

7. Load

Purpose: To recall a source file from disk or cassette into the editor.

Command: CBM L

Sequence: Hold down the CBM key and press 'L', type the name of the file to be recalled and press RETURN.

Notes: The source file currently held in memory will be lost if it has not been previously saved, a warning message will be displayed if the file has not been saved in its present form.

Cassette users should reply to the 'FOUND' message by pressing the CBM (or CTRL) key.

Disk users may use the '*' pattern matching symbol when loading a file into the editor (e.g. to load a file called 'document', type 'doc*' onto the command line, and the first file on the disk starting with 'doc' will be loaded into memory). However, an attempt to save the file with the same abbreviation will not be permitted.

8. Move

Purpose: To highlight a range of lines in one part of the source file and move it to another location in the same source file.

Command: CBM M

Sequence: Position the cursor on the first line of the range to be moved, hold down the Commodore key and press 'M'. Highlight the source lines to be moved and press RETURN. Now move the cursor to the beginning of the new location where the highlighted source is to be moved to, and press RETURN.

Notes: Lines are highlighted with the CRSR up, CRSR down, F3 and F4 keys. The move will not take place if there is insufficient memory for a duplicate copy of the highlighted lines to be kept during the move. Move the source lines in smaller sections if short of memory.

9. New File

Purpose: To delete all source file lines and symbol table from memory, to allow a new file to be started.

Command: CBM N

Sequence: Hold down the CBM key and press 'N', type a 'Y' to clear the memory, or press the STOP key to cancel the command.

10. Old File

Purpose: To restore the source file just cleared from memory by a New File (CBM N) command.

Command: CBM O

Sequence: Hold down the CBM key and press 'O'.

11. Print/List

Purpose: To list the source file to the screen and, optionally, out to a printer.

Command: CBM P

Sequence: Hold down the CBM key and press 'P'. To list to the printer press 'P' once again. To start the listing, press RETURN. To halt the listing press STOP.

Notes: The listing starts at the current cursor location and continues to the end of the source file.

The Assembler directives .LIST, .NOLIST, and .PAGE, that control assembly listing, are ignored.

The progress of the listing may be seen from the position of the white triangle along the memory scale.

EDNA automatically skips over paper perforations, assuming that the paper length is 11 inches, with 1/6" line spacing. A blank line in the last five lines of a page initiates a page skip. This is designed to improve the presentation of the printed source file, as it will reduce the number of routines that are split over two different pages. Page skips can be forced by putting extra blank lines into the source file.

12. Query - Programmer's Calculator

Purpose: To evaluate and display an expression in its decimal and hexadecimal equivalents. To show the address of a label, after assembly.

Command: CBM Q

Sequence: Position the cursor on the expression in the source file that is to be evaluated. Hold down the CBM key and press 'Q'. Characters from the cursor position up to the next space are then automatically placed on the command line. To enter your own expression or calculation - type over these characters. Press RETURN to display the result of the expression. Press STOP to abandon the query command.

Notes: Labels may be included in expressions, but **only after the source file has been assembled**.

The expression may include any of the forms permitted for an operand: decimal; hexadecimal; binary; ASCII (text in quotations); program counter; labels; arithmetic operators (+,-,*,/).

For example:

```
3*$90
END-START
'A'
```

13. Replace

Purpose: To search for a sequence of characters within the source file and to, selectively, replace it with another sequence.

Command: CBM R

Sequence: Place the cursor on the first line to be searched, hold down the CBM key and press 'R', type the sequence of characters to be searched for and press RETURN. Type the new sequence of characters, and press RETURN.

When a match is made: the line is displayed in amended form. Press STOP to restore the original line. The amended line enters the source file as the cursor is moved off the line, Resume the search by pressing SHIFT and RETURN together.

If the modified line no longer conforms to the correct editor syntax you should correct the line before proceeding with the REPLACE command or further editing operations. Press STOP twice to abandon the command.

14. Save

Purpose: To save the source file held in memory to either cassette or disk (as selected upon entering the editor for the first time, or by using the CBM 1 command).

Command: CBM S

Sequence: Hold down the CBM key and press 'S', type the filename and press RETURN.

Notes: If you have previously loaded or saved a source file the known filename is prompted.

To REPLACE a file already on the disk, insert '@0:' before the filename. The Load command ignores this prefix.

Press STOP instead of RETURN to abandon the SAVE command.

If you wish to change the name of the source file in memory, issue the LOAD or SAVE command, type the new filename, then press the STOP key.

15. Mark Position

Purpose: To mark the current position in the source file. It can then be 'jumped-to' at any time using the 'Go To Marked Position' command.

Command: CBM X

Sequence: Position the cursor to the source line where the 'mark' is to be set, hold down the CBM key and press 'X'.

Notes: This command, together with the CBM Y (Go To Marked Position) command, allows a quick return to the working location, after browsing through other parts of the source file.

16. Go To Marked Position

Purpose: Moves the cursor to the position in the source file as set by the CBM X command.

Command: CBM Y

Sequence: Hold down the CBM key and press 'Y'.

17. Append

Purpose: To delete all source file lines starting from the current cursor location, and to then append a source file from disk or cassette.

Command: CBM Z

Sequence: Place the cursor after the last line to be retained in the current source file. Hold down the CBM key and press the 'Z' key, type the name of the file to be appended and press RETURN.

Notes: It is recommended that you SAVE your source file before using the APPEND command.

18. Set Foreground Colour

Purpose: To change the colour of the text and the cursor on the screen.

Command: CBM 1

Sequence: Hold down the CBM key and press '1'; repeat this process until the required colour is selected.

19. Set Background Colour

Purpose: To change the background colour of the screen.

Command: CBM 2

Sequence: Hold down the CBM key and press '2'; repeat this process until the required colour is selected.

20. Set Border Colour

Purpose: To change the colour of the border around the screen.

Command: CBM 3

Sequence: Hold down the CBM key and press '3'; repeat this process until the required colour is selected.

21. Default Colours

Purpose: To change the screen colours back to the default startup colours.

Command: CBM 4

Command: Hold down the CBM key and press '4'.

Chapter 3.

The Assembler

This Chapter details the program syntax used by the Assembler, and the special assembler **directives** that cause certain actions to be taken at assembly time. The syntax of instructions adheres closely to the MOS-Technology specification, already familiar to users of the Commodore assembler.

Constants

Constants are fixed numerical values. They may be expressed in the following forms:

Decimal (base 10)	A number without any prefix will be interpreted as decimal. Bytes may hold values 0 to 255, two bytes (a word), the values 0 to 65535.
Hexadecimal (base 16)	Prefixed by a '$' sign, a byte may hold a value of $00 to $FF and a word the value $0000 to $FFFF. Unlike decimal each hex digit relates to 4 specific bits (a nybble) of a byte, so the binary pattern is easily obtained.
Binary (base 2)	Prefixed by a '%' sign, a byte may hold the value %00000000 to %11111111. Although a word may be expressed in binary, the hexadecimal form is usually more convenient.
Ascii or Text	The Assembler evaluates any character enclosed by single quotes as the ASCII code for the character. For example: 'A', 'a', '$', etc. Use two single quotes to denote the ASCII code for a single quote.

The following, expressed in each of the above forms, will have the same assembled value:-

65 $41 %01000001 'A'

Labels

Labels are names typed at the start of a line in the source file. A label can be used to represent a program address, variable or constant. A label should offer some description of its purpose, to make the program more easily understood. The use of labels, rather than absolute addresses, adds substantial flexibility when altering the program.

Labels may be up to 6 characters long and should start with a letter of the alphabet in upper or lower case. A label **must not** contain commas, brackets, spaces or arithmetic operators, such as '+', '-', '*' or '/'.

Some examples of labels:

```
LABEL   =$2000
label   STA   CHAR
A1000   ADC   #<1000
ST.REG  STX   LOW
CIOUT   JMP   $FFD2
CHAR    .BYT  0
LOW     =$32
```

Expressions

Constants and labels can be expressed in a variety of ways using the following arithmetic operators:

+	addition
-	subtraction
*	multiplication
/	division

Unlike BASIC, where multiplication takes precedence over addition, and brackets may be used to alter the order of precedence, the Assembler evaluates expressions strictly from left to right. The result and any intermediate calculation will be rounded down to a 16-bit integer. When in doubt, use the **query** command (see Chapter 2 - The Editor - page 2-10) to try out an expression.

Some example expressions:

```
Z'+1
LABEL/2
LABEL1-LABEL2
*-LABEL
*+1
```

Beginner's Note:

Remember that the expression is evaluated at ASSEMBLY TIME, and that the calculations use the **address** of a label, **not** the contents of memory addressed by that label.

Low And High Byte Operators

As the 6502 microprocessor uses 8-bit registers, it is often necessary to break down 16-bit values into their constituent high and low bytes. The operators for this are:

<	low byte
>	high byte

For example, to place the address of the screen video RAM in a zero page 'address pair':

```
P1      = $02
SCREEN  = $0400

        LDA  #<SCREEN       ;$00 - LOW BYTE
        STA  P1
        LDA  #>SCREEN       ;$04 - HIGH BYTE
        STA  P1+1
        RTS
```

Note: These operators may be used with an expression, and will operate on its result. For example, >$1234-1 = $12, <$1234-1 = $33.

Source Line Format

Each source line must conform to the following format:

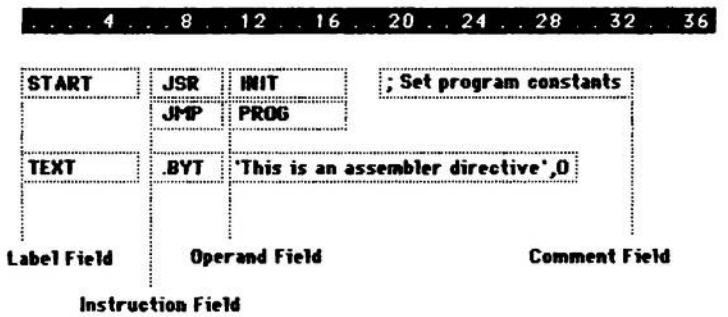

Figure 3.1 Source Line Format.

1. THE LABEL FIELD

This is the area from the **first column** of a row of text up to the instruction field. Labels should appear in this field once only, to define a particular **memory address**. Labels are optional and are only required where there are references made to it.

2. INSTRUCTION FIELD

The instruction field contains the three character instruction op-code mnemonic or an assembler directive. When typing in a line without a label: type a space or press RETURN (to tab), before typing in the instruction.

3. OPERAND FIELD

The operand specifies the addressing mode and the register, data or address to which the instruction applies. Where the addressing mode is implied in the instruction no operand is required (e.g. PHA). The operand should be separated from the instruction by one space.

4. COMMENT FIELD

Type a semi-colon followed by the comment. Comments may also be entered as headings across the whole line. We recommend the use of comments at the start of each routine to summarise its function, detailing entry and exit conditions. Within a routine comments should point out each process and conditions for branching. Comments should be added to specific instructions where their use is not self evident, such as forced (unconditional) branches, ingenious use of the flags, etc. Comments are ignored at assembly time, but appear in assembly listings.

Appendix 1 gives examples of valid 6502 addressing modes recognised by this assembler.

Assembler Directives

An **assembler directive** is an instruction **to the assembler** to perform a specific task when assembling a program. Directives fall into two categories, those that **generate object code** and those that **control assembly**.

1. DIRECTIVES THAT GENERATE OBJECT CODE

There are three directives that generate object code:

. BYT

This directive generates bytes of **data** within a program. Any attempt to assign a value greater than $FF (255) will produce an error at assembly time. Several bytes may be defined on one line by separating them with commas.

```
e.g.    .BYT    $20,100,'Z'+1,LABEL,<$1234,>$1234      - 6 Bytes.
```

Place single quotes around text to define a 'string' of ASCII data.

```
e.g.    .BYT    Text terminated by zero',0             - 24 Bytes.
        .BYT    Text terminated by setting bit',7'+$80 - 31 Bytes.
```

.WOR

This directive generates 2 byte words in low, high byte order.

e.g. .WOR $FFFF,$1234,12345,LABEL - 8 Bytes.

.DBY

This directive generates 2 byte words in high, low byte order. This is rarely used.

2. DIRECTIVES THAT CONTROL ASSEMBLY

Equates

LABEL = VALUE (or EXPRESSION)

Labels may be **equated** to constants, other labels or expressions. A label can only be equated in terms of another label if it has **already been defined in the program**. For example:

```
LABEL1      =10000
LABEL2      =%10101010+'z'-1+LABEL1*3/2
CHROUT      =$FFD2
```

Set Program Counter

* = address

This directive is used to set the address at which the program should start. The Assembler will increment its 'program counter' from this point as it assembles the object code.

Allocate Memory

```
* = * + constant
```

This directive increments the 'program counter' by the constant value. This is normally used to allocate memory for variable storage. For example:-

```
                *-$02
TEMP1           *=*+1
TEMP2           *=*+1
P1              *=*+2
```

This memory has an initial value of $FF in each byte allocated.

Assemble To Memory

```
.MEM
  source code
.NOMEM
  source code
```

During assembly, these directives can be used to store selective sections of assembled machine code directly into memory.

List Assembly

```
.LIST
  source code
.NOLIST
  source code
```

During assembly, these directives can be used to selectively print parts of an assembly listing.

Disk Load Address

```
@ = address
```

This directive allows the program load address to be independently set. Normally the assembled program will be loaded back at its original program counter address (*=address). The program load address is stored in the first two bytes of the assembled machine code file.

Linked Files (Disk or Cassette)

```
.FIL  next filename
.END  first filename
```

Files may be 'linked' together to effectively form one large source file at assembly time. The last line of each source file contains the call to the next (**.FIL** filename). The last file must contain a **.END** directive that specifies the name of the first file in the list (**.END** filename). Assembly therefore starts and ends with the first file loaded in memory. Each file is loaded into memory twice during assembly: once for pass 1 and again for pass 2.

Library Files (Disk only)

```
.LIB  filename
```

Unlike linked files, library files are **not** loaded into memory but are read directly from the disk. At the end of the library file control is returned to the file in memory. The file in memory need only be a series of **.LIB** calls in an extreme case. Very large programs may be assembled in this way, the only limit being when the entire memory is taken up by the symbol table (very unlikely!). Library files must not contain the **.FIL** or **.END** directives.

Library files are just ordinary EDNA source files. One of the most practical uses for a library file is a complete definition of all 'operating system' addresses. This 'library file' can then be **.LIB**'bed into the front of any source file. This saves considerable time, effort and errors !

```
.IF  expression
     source code
[.ELSE]
     source code
.ENDIF
```

Where several different versions of a program are required, perhaps to run on different computers, the assembler can be made to 'skip' over sections of the source file and assemble alternative sections determined by an equated value at the start of the program. The equated value will be judged **true** if it is **not zero** (<>0), **false** if it is **equal to zero** (=0).

If, at assembly time, the label or expression after the **.IF** directive is found to be true the source lines following will be assembled: if it is false they will be ignored until the next .ELSE or .ENDIF directive.

The **.ELSE** directive is an optional part of the conditional assembly directive, and is used after the source lines that follow a .IF.

The **.ENDIF** directive turns conditional assembly 'off' and assembly continues as normal.

Some examples of conditional assembly:

```
C64       =1

          .IF C64        ; Is C64 true (i.e. C64=1) ?
SCREEN    =$0400         ; Yes - so compile
SWIDTH    =40
          *=$C000

          .ELSE          ; Is C64 false (C64=0) ?
SCREEN    =$8000         ; No - so don't compile
SWIDTH    =80
          *=$7000

          .ENDIF         ; End conditional assembly - compile
                         ; everything after this
```

The above example details how values for the screen memory, screen width and program memory might be set up for a Commodore 64 as opposed to a Commodore 8000.

End Assembly

```
.END
```

This directive ends the assembly of the source file. It can be placed anywhere in the source file and may therefore be used to produce a partial assembly of a much larger source file.

After Assembly

One of the most useful commands is QUERY. After an assembly, QUERY will show the assembled address, or value, of any label: both equated labels and labels within program source routines.

Chapter 4.

The Monitor

The machine code monitor makes machine code programs easier to test and get working! When you first switch on your computer with the cartridge inserted, EDNA 'power's up' in the Monitor. Exit to BASIC by typing 'x' and RETURN, press RUN to enter the Editor.

Monitor Command Summary

Abbreviations Used:

"filename"	Name of the file up to 16 characters in length. Quotes may be omitted if there are no embedded spaces in the filename.
[ ]	Square brackets indicate optional sections of a command.
xxxx	Start address in hexadecimal.
yyyy	End address in hexadecimal. Commands honour this end address, unlike many other monitors.
zzzz	Offset address in hexadecimal.
[n]	Device number. '1' for cassette, '8' for disk. Defaults to '8' if omitted.
[printer]	Printer type. 'S' for serial Commodore printers, 'P' for centronics parallel, 'PL' for parallel printers with requiring linefeeds. Defaults to screen output.

All command parameters can be separated by either a space or a comma. Type the command, plus parameters and press RETURN to execute it.

Remember, the display commands, such as Disassemble and Memory are abandoned by pressing SHIFT and RETURN.

Commands:

C xxxx yyyy zzzz [printer]	Compares the block of memory from xxxx to yyyy with the block of memory starting at zzzz. Byte mismatches are shown, along with their addresses, on separate screen lines. If the alternate character set is selected (SHIFT CBM): the first block is taken from all RAM and the second block from the normal memory map. This allows RAM and ROM at the same addresses to be compared. STOP cancels the compare.
D xxxx	Disassembles one line at this address. Move the cursor up and down the left margin to scroll the disassembly through memory.
D xxxx yyyy [printer]	Disassembles memory from xxxx to yyyy. Press CTRL to pause.
F xxxx yyyy aa	Fills the block of memory from xxxx to yyyy with the byte of data aa.
G xxxx	Executes the machine code program starting at address xxxx. This command is used as a 'subroutine' from within the monitor. EDNA initialises the stack pointer to $FD, with the top two bytes of the stack containing a return address back to the Monitor.

H xxxx yyyy aa bb [etc.] [printer]	Hunts memory between xxxx and yyyy for all occurrences of aa bb (etc.). The address and following 8 bytes of data are displayed for each occurrence of the specified data. STOP cancels the hunt.
H xxxx yyyy "text" [printer]	As above, but hunts for "text".
L "filename" [n] [xxxx]	Loads the named file into memory. If a load address is not specified the file loads at the address specified at the start of the file. Cassette files originating from other programs may contain an ID byte that prevents them from being loaded to the supplied xxxx address.
M xxxx	Displays one line of memory at xxxx. Move the cursor up and down the left margin to scroll through memory.
M xxxx yyyy [printer]	Displays memory from xxxx to yyyy. Press CTRL to pause.
R	Register display. These registers are displayed automatically when a 'BRK' instruction is encountered or during 'Walk' operation. The registers are shown in hexadecimal, with the exception of the flag register, which shows the binary value of individual flags.
S "filename" [n] xxxx yyyy [zzzz]	Saves memory contents from xxxx to yyyy as a program file (add 1 to the end address for cassette). A different 'reload' address zzzz (disk only) may be specified. If the alternate character set is selected (upper/lower case) the file will be saved from the RAM underneath ROM (disk only).

T xxxx yyyy zzzz	Transfers a block of memory starting at xxxx and ending at yyyy to the memory location starting at address zzzz. Memory transfer can be from all RAM if the alternate character set is selected.
V "filename" [n] [xxxx]	Verifies a file on disk or cassette with the file in memory. A '?' is displayed if the two differ. Closer inspection of differences may be made by loading the file to an offset address and using the **compare** (C) command.

W xxxx yyyy zzzz

Walks through code. This command can be used to trace through a program starting at xxxx. The parameters yyyy to zzzz are used to specify a range of addresses where EDNA should 'single-step' each instruction. Outside this range the program will run continuously, though at considerably reduced speed. When the program passes through the specified range the registers are displayed, the next instruction to be executed is disassembled and listed to the screen.

CRSR down Continues the trace.

SHIFT-W Resumes the trace mode.

SHIFT-C Cancels trace mode and resumes the program at normal speed.

No method of tracing is infallible, here are a few don'ts:

1. Don't attempt to single-step through any part of the operating system used by the trace routine itself.
2. Don't trace through any routines where time is a critical factor, such as disk I/O.
3. Don't trace through any programs that redefine the screen modes or location.
4. Don't single-step through routines reading input data from the screen.
5. Don't trace through routines which 'page out' the operating system.

X	Exits to BASIC (protect file - see page 1-2)
SHIFT X	Exits to BASIC (don't protect - see page 1-2)
/	Displays disk status.
>[S0:FILE* etc.]	Sends disk commands. See also the disk command in Chapter 2 - The Editor.
$[0:FILE* etc.]	Displays a disk directory, selective directory etc.
PGC .A .X .Y SP NV. BDIZC *FFFF FF FF FF FF 00100000	The displayed registers may be edited on the screen.
:xxxx aa bb cc dd [etc.]	Write data aa to address xxxx, bb to xxxx+1, etc.
.xxxx aa bb cc [etc.]	Write data as above and disassemble the instruction.
BRK instruction	A "BRK" instruction **within a program** returns control to the Monitor **regardless of the value of the stack pointer**. The value of the program counter, A, X, Y, the stack pointer and status registers are displayed.
RTS instruction	Unless the program running resets the stack pointer, an RTS instruction will return control to the machine code Monitor. The Register command can be used to show A, X, Y, the stack pointer and status registers.

The 'M' and 'D' commands produce a listing compatible with the ':' and '.' data entry commands. This allows inspection and modification of memory.

The Promenade C1 Eprom Programmer

Commands have been included to enable the Promenade Eprom Programmer to be used within the Monitor. This unit will program all types of Eprom, using various programming methods as selected. The commands included follow the same format as in the Promos software included with the unit with the exception that the commands are in **hexadecimal** rather than decimal. The error light stays on steadily after an error has occurred rather than flashing.

Z	Initialise the system.
↑I xxxx yyyy zzzz [CW] [PMW]	Program the memory from xxxx to yyyy into the Eprom at address zzzz. CW controls the Eprom type, PMW defines the programming method. If the alternate character set is selected the Eprom is programmed from all RAM. Note that the CW and PMW must be entered in hexadecimal.
£ xxxx yyyy zzzz [CW]	Read Eprom into memory.
SHIFTed E	Erase 48016P Earom.

Memory Used By The Monitor

Zero page addresses $02 - $2E	Workspace used by the Monitor. Machine code programs may also also use any of this area as no long term data is stored here. This area is pushed onto the stack during single-step tracing, so that its use remains transparent to the program being traced.
$0100 - $011F	Workspace.
$0120 - $012A	Pseudo-registers.
$02FF	Flag to indicate memory mode - protect or don't protect files.
$0300, $0301	Vector for BASIC.

Memory Swapped When Entering Or Leaving BASIC

$02 - $8F	with $A072 - $A0FF
$0B00 - $26FF	with $A100 - $BFFF
$2700 - $56FF	with $D000 - $FFFF

Chapter 5.

Using Machine Code Within A BASIC Program.

Getting To BASIC From The Monitor

EDNA allows you to develop both machine code and BASIC programs in memory at the same time. Two different methods of using BASIC are provided: PROTECT SOURCE (press X and RETURN) and DON'T PROTECT SOURCE (hold down the SHIFT key and press X, then RETURN).

PROTECT SOURCE allows both BASIC and Assembler files to be protected in memory at the same time. You can then swap between them at will. The amount of memory available is restricted to 20223 bytes from BASIC. BASIC program source and variables are preserved when using the Editor, similarly Editor source is preserved when using BASIC. Get back to the monitor with a SYS 57000 call from BASIC.

DON'T PROTECT SOURCE allows the full 38911 bytes of BASIC memory to be used. BASIC and Editor source files are not preserved when switching between the Editor and BASIC. Get back to the Monitor with a SYS 57000 call from BASIC.

To get to the machine code Editor from the Monitor, press the RUN key.

This chapter gives details of how to make the best use of the available memory when developing a program in both BASIC and machine code.

Zero Page Locations

BASIC uses all the zero page locations from $02 to $8F at some time or another, and the operating system uses most of the rest. Although there are a few locations it is safe to use, the safest solution is to push an area of zero page used by BASIC onto the stack at the start of a machine code call and replace it on exit from the routine.

Workspace Available

A section of filespace may be reserved as workspace. In addition the following locations are available:

$02A7 - $02FE	(The Monitor uses $02FF)
$0334 - $033B	
$033C - $03FB	(Except when using a cassette)
$03FC - $03FF	
$07E8 - $07F7	(The screen only uses 40 * 25 bytes)

Communication between BASIC And Machine Code

Parameters POKEd into memory within BASIC can be read by a machine code program, modified where necessary and subsequently PEEKed back from BASIC. BASIC also loads the processor registers from stored locations at the start of a SYS call and stores the return values back at the end of the call. These locations are:

A	$030C	(780)
X	$030D	(781)
Y	$030E	(782)
STATUS	$030F	(783)

Except when set to zero at switch-on, these locations are otherwise untouched. Note, that if $030F (status register storage) becomes corrupted it is possible for the processor to be set in its decimal mode, with unpredictable results.

How And Where To Store The Machine Code Program

During development of a program that incorporates both machine code and BASIC it is probably easiest to assemble the machine code to the 4k memory block starting at $C000. This area is not used by BASIC or the Assembler/Editor and it is accessible at all times from both.

For, the finished product, the following two alternatives are suggested to save the user from the need to load several files:

1. Download machine code files within the program.
2. Append machine code to the end of a BASIC file.

1. Download Machine Code Files Within BASIC

Files may be loaded into memory and called as required during the program, in which case they may all share the same memory, or alternatively, all the files may be loaded when the program is first run and called whenever necessary. The $C000 area of memory remains the first choice of position for machine code in memory.

As an alternative, you may change the top of memory pointer to allow extra room at the top of the filespace from $9FFF downwards (the pointer normally points to $A000). If BASIC is entered via the Monitor using the X command (share memory between the Editor and BASIC) this limit is lowered to $56FF. Where you change the top of memory pointer, POKE the pointer down at the **start** of a program, otherwise strings are liable to overwrite the machine code. Call **CLR** and BASIC will initialise the string pointers to the new top of memory address. In the example on the next page, the following line would be inserted after line 10.

20 POKE 55, *low-byte* : POKE 56, *high-byte* : CLR

One snag with downloading files is that it causes BASIC to return to the start of the program. Fortunately, the variables remain intact - so you can produce a program of the form:

```
10 ON A GOTO 1010,2020 etc.
30 REM initially A=0 so the program
40 REM will drop through to here
50 REM after executing line 1000
60 REM the program will return
70 REM to line 1010 etc.

1000 A=1: LOAD "file-1",8,1
1005 REM program never reaches this line
1010 REM it returns to this line

2000 A=2: LOAD "file-2",8,1
2010 SYS start-address
```

2. Append Machine Code To A BASIC Program

This method allows machine code to be saved and loaded as part of a BASIC program.

As an indication of the end of the program the BASIC command interpreter detects two NULLs forming what would otherwise be the pointer to the next line. The BASIC text editor maintains a separate pointer which is set during a cold start (switch on machine or SHIFT X Monitor command) to the first free byte after the NULLs. This pointer is used as the end of file pointer when the program is saved, and the starting point for variable storage which accumulates above the program while it running. In normal use the pointer and NULLs are moved up and down memory together as the program is edited. If machine code is placed after the end of the BASIC program, and the end of file pointer is moved up to accomodate, the program will run quite normally. However, **the BASIC program should not be edited in this form**, the machine code would be relocated as it was moved about in memory by the BASIC editor.

To append a file in this way:

a) Enter BASIC from the Monitor (X command) and load the BASIC file. It is likely that the program already contains SYS calls to the area in memory where the program was developed. Change all of these calls to **SYS 45640**, the new addresses are not yet known. Using a five digit 'dummy' call ensures that the program does not increase in size when the final values are substituted; it does not matter if it shrinks by a few bytes. The address 45640 is the ?ILLEGAL QUANTITY ERROR call, in case you forget to change them later.

b) Note the end address of the program using the statement **directly** (without a line number) from BASIC:

```
?PEEK(45)+256*PEEK(46)
```

c) Exit BASIC (SYS 57000) and enter the Editor (RUN key). Load the assembly code source file, if it is not already present.

d) Change the start address to be greater than or equal to the end of BASIC address obtained in step b) (with the '* =start-address' directive). It is advisable to leave a small gap between the two to allow small changes to be made to the BASIC program without the need for reassembly. Do **not** edit the BASIC program once the machine code is incorporated.

e) For disk users: **assemble** the program **to the disk** with the new start address. Note the filename.

For cassette users (or as a short cut): If the end of the machine code falls below $26FF it is possible to assemble the object code directly into the area of memory where the BASIC file is stored while within the Monitor or Editor. The memory from $0800 to $26FF is swapped with RAM at $A100 to $BFFF (nearly 8k). The remainder of the swapped file space is from $D000 to $FFFF.
The object code is assembled to memory using the offset ⊕ -*+$9900 placed on the line after the *-start-address directive.

f) Use the **Query** command to note the decimal value of all entry addresses used by SYS calls. Note also the high and low byte values of the end address by querying:

<* *low-byte*
>* *high-byte*

g) Exit the Editor using the RUN key and enter BASIC (X command). Substitute the real values into the SYS calls noted above. If the object code was assembled to disk, load it back into memory:

LOAD "*filename* ",8,1

Update the end of file pointer to include the machine code section obtained from the values obtained in f) above.

POKE 45, *low-byte* : POKE 46, *high-byte*

From now on you should **not** edit the program.

h) Save the file to cassette or disk.

SAVE " *filename* " (cassette)
SAVE " *filename* ",8 (disk)

i) RUN the program to test it.

The Monitor X command allocates a BASIC file area from $0800 to $56FF. If your files are too large for this mode the SHIFT X command may be used. This allocates the full file area to BASIC from $0800 to $9FFF, overwriting assembler source files. Similarly, assembler files overwrite BASIC files. If this mode is used files **may not be assembled to memory** as described in section e), but must loaded from disk as required.

Appendix.

A1. 6502 Addressing Modes

The following are some examples of valid 6502 addressing modes. An address may be a constant, label or expression.

Implied addressing, 1 byte

```
CLC
PHA
```

Accumulator Addressing, 1 byte

```
ASL A
ROR A
```

Immediate Addressing, 2 bytes

```
LDA #value
ORA #%01010101
```

Relative Addressing, 2 bytes

Forward and backward branches of up to half a page are allowed. The assembler will calculate the relative address and reject any out of range values as errors.

```
BNE address
BEQ address
BCC address
BCS address
```

Zero Page and Absolute Addressing, 2 and 3 bytes

The Assembler will choose the zero page mode where possible.

```
LDA address
JSR address (absolute mode only)
JMP address (absolute mode only)
```

Zero Page and Absolute Indexed Addressing, 2 and 3 bytes

```
LDA address,X
LDA address,Y
```

A2. Extra Instructions For The 65C02 Microprocessor

Push and Pull Index Registers

```
DA          PHX
5A          PHY
FA          PLX
7A          PLY
```

Increment and Decrement Accumulator

```
1A          INC  A
3A          DEC  A
```

Unconditional Branch

```
8002        BRA  *+2
```

Store Zero In Memory

```
64FF        STZ  $FF
9CFFFF      STZ  $FFFF
74FF        STZ  $FF, X
9EFFFF      STZ  $FFFF, X
```

Test And Reset Bits

```
04FF        TSB  $FF
0CFFFF      TSB  $FFFF
14FF        TRB  $FF
1CFFFF      TRB  $FFFF
```

Extra Bit Test Modes

Note: N and V flags are not affected during immediate instruction

```
89FF        BIT  #$FF
34FF        BIT  $FF, X
3CFFFF      BIT  $FFFF, X
```

Indexed Indirect Jump

```
7CFFFF      JMP  ($FFFF,X)
```

Indirect Addressing (not indexed)

```
72FF        ADC  ($FF)
32FF        AND  ($FF)
D2FF        CMP  ($FF)
52FF        EOR  ($FF)
B2FF        LDA  ($FF)
12FF        ORA  ($FF)
F2FF        SBC  ($FF)
92FF        STA  ($FF)
```

A3. Parallel Printer Connections

Printer		User Port	
Pin	Function	Pin	Name
1	Strobe	M	PA2
2	Data 0	C	PB0
3	Data 1	D	PB2
4	Data 2	E	PB3
5	Data 3	F	PB4
6	Data 4	H	PB4
7	Data 5	J	PB5
8	Data 6	K	PB6
9	Data 7	L	PB7
10	Acknowledege	B	FLAG
Pins 11-18 unused.			
19	Ground	1	Ground
20	Ground	1	Ground
21	Ground	1	Ground
22	Ground	A	Ground
23	Ground	A	Ground
24	Ground	A	Ground
25	Ground	N	Ground
26	Ground	N	Ground
27	Ground	N	Ground
28	Ground	12	Ground
29	Ground	12	Ground
Pins 30-36 unused.			

You may purchase a cable in this configuration by contacting VIZA SOFTWARE at the address given below.

A4. Memory Used By EDNA

$02-$8F	Zero page scratchpad. Initialised on entry to EDNA so it may be used freely from outside.
$02A8-$02FE	Assembly buffer (also freely useable).
$0800-	Buffer space.
$08CC-	Filename store.
$08EE,F	Old end of file pointer after NEW.
$08F0, 1	Pointer to start of source file.
$08F2, 3	Pointer to end of source file.
$08F4, 5	Pointer to the absolute limit of the symbol table (normally equal to end of source pointer, but holds value of largest source file in a linked file assembly).
$08F6, 7	Pointer A, end of current symbol table.
$08F8, 9	Pointer B, start of symbol table for Pass 1.
$08FA, B	Pointer C, start of symbol table for Pass 2 ($A000).
$0900	Zero byte
$0901-$A000	Source file extends upwards, symbol table extends downwards.

Suggestion: it is possible to 'carry over' a symbol table from one assembly to another by copying pointer A into pointer B, using the Monitor. Do this after each assembly, and all symbol tables will then be appended. Use (or don't use !) the NEW command to reset these pointers back to their usual value.

A5. Source File Format

All lines terminate in a zero byte, a blank line is just a zero byte. All 6502 mnemonics and EDNA directives are tokenised. All tokens have their top bit set. So it can be assumed that if the first byte in a line has its top bit set then the line has no label. If the top bit is clear, then the byte is the first character of a label.

Comments are compressed by omitting single spaces and setting the top bit of the next character. Double spaces are stored as one space with the top bit set. All other characters are stored in ASCII.

Source lines can be inspected using the Monitor. Source lines are normally stored starting at location $0901.

List of tokens

$80	ADC	$81	AND	$82	ASL
$83	BCC	$84	BCS	$85	BEQ
$86	BIT	$87	BMI	$88	BNE
$89	BPL	$8A	BRA*	$8B	BRK
$8C	BVC	$8D	BVS	$8E	CLC
$8F	CLD	$90	CLI	$91	CLV
$92	CMP	$93	CPX	$94	CPY
$95	DEC	$96	DEX	$97	DEY
$98	EOR	$99	INC	$9A	INX
$9B	INY	$9C	JMP	$9D	JSR
$9E	LDA	$9F	LDX	$A0	LDY
$A1	LSR	$A2	NOP	$A3	ORA
$A4	PHA	$A5	PHP	$A6	PHX*
$A7	PHY*	$A8	PLA	$A9	PLP
$AA	PLX*	$AB	PLY*	$AC	ROL
$AD	ROR	$AE	RTI	$AF	RTS
$B0	SBC	$B1	SEC	$B2	SED
$B3	SEI	$B4	STA	$B5	STX
$B6	STY	$B7	STZ*	$B8	TAX
$B9	TAY	$BA	TRB*	$BB	TSB*
$BC	TSX	$BD	TXA	$BE	TXS
$BF	TYA	$C0	'='	$C1	'*='
$C2	'@='	$C3	'.BYT '	$C4	'.WOR '
$C5	'.DBY '	$C6	'.FIL '	$C7	'.END '
$C8	'.LIB '	$C9	'.IF '	$CA	'.ELSE '
$CB	'.ENDIF '	$CC	'.END'	$CD	'.MEM'
$CE	'.NOM'	$CF	'.LIS'	$D0	'.NOL'
$D1	'.PAG'	$FF	';' (start of comment)		

* 65C02 instructions.

A6. Messages From EDNA

Symbol Table Overflow

As EDNA assembles the source file, it builds a list of defined labels. This list of labels is known as the symbol table and is stored in the unused part of memory. If the source file is particularly large there will be insufficient room to hold the complete symbol table. The source file should be split into two separate, smaller files. The second source file can then be **.LIB**'ed at the end of the first file. Alternatively, use the **.FIL** directive to chain from the first file to the second.

Symbol Value Error

A forward reference, that itself contains a forward reference cannot be evaluated correctly. Check the use of the label.

Unresolved Expression

The expression has an incorrect syntax and cannot be fully evaluated.

Label Not Defined

A referenced label has not been defined anywhere in the source file.

Syntax Error

The 6502 instruction or assembler directive is incorrect.

Out Of Range

The result of an expression has a value larger than the 6502 instruction can process.

Invalid Addressing Mode

The combination of memory location, registers and 6502 instruction is not supported by the 6502 or 65C02 processors.

Corrupted Source

EDNA has detected a format error in the source file. This can be caused by the user's program writing to the RAM area that EDNA uses to hold the source file. Take care, save the program source before testing the program.

Program Counter Error

The internal 'program counter' used by the assembler has been set 'backwards' during assembly.
This is due to incorrect use of the '*-' directives.

End Of Assembly

EDNA has completed its assembly of the source file into executable 6502 machine code. If error messages have been displayed, correct the errors and re-assemble. Only execute the assembled program when it is error free.

A7. Sample Disk Programs

On the supplied disk we have included a conversion program that will convert Commodore assembler source files and Mikro Assembler source files to EDNA format. *This program is called* 'CONVERT TO EDNA' and should be loaded and run from BASIC. Full instructions are given when running the program. Note that long labels used in a Mikro source file are truncated. Any source that cannot be converted is turned into a comment.

For beginners, and in case of difficulty, we have included several sample source files. These cover the Commodore Kernal routines, zero page use, screen handling, printing and disk file operations. By using these examples, you should soon be 'off and running'.

If you create source files that you think would be of benefit to other beginners, we would be pleased to put them on this disk. The sample files provide are called 'ZERO.ED', 'SCREEN.ED', 'DISK.ED' and 'PRINTER.ED'. The file called 'KERNAL' is a list of the Commodore Kernal routine addresses. This can be .LIB'ed into your own source files.

A8. User Registration Service

Please register now ! Registered users are notified of updates to the product, this includes additional features as well as detailing any deficiencies in the version purchased by the registered user. This is also the medium for any suggestions you may have.

To become a registered user just complete and return the registration slip enclosed. Please enclose your name, address and cartridge serial number (on the back of the cartridge). Send to:

EDNA Registration,
VIZA SOFTWARE Ltd,
CHATHAM HOUSE,
14, NEW ROAD,
CHATHAM
KENT. ME4 4QR ENGLAND.

Please let us know if you have any suggestions for improving this product. If you have any problems using EDNA you should first contact your dealer. If your dealer is not able, or is unwilling to help, then please WRITE to us detailing the exact nature of the problem. Enclose your full name, address and telephone number. We will endeavour to answer your problem as quickly as possible. We are not able to enter into discussions about your use of 6502 machine code.

Index.

A

B

C

D

F

G

H

I

L

M

S

T

U

V

W

X

Y

Z

Monitor Commands

Abbreviations used:

"filename"	Name of the file up to 16 characters in length.
[ ]	Sections of a command that may be omitted.
xxxx	Start address in hexadecimal.
yyyy	End address in hexadecimal.
zzzz	Offset address in hexadecimal.
[n]	Device number. '1' for cassette, '8' for disk.
[printer]	Printer type. 'S' for serial, 'P' for parallel, 'PL' for parallel with line feeds.

Commands:		Page/ref.
C xxxx yyyy [printer]	Compare memory	4-2
D xxxx	Disassemble one line	4-2
D xxxx yyyy [printer]	Disassemble memory	4-2
F xxxx yyyy aa	Fill memory with data aa	4-2
G xxxx	Execute program from xxxx	4-2
H xxxx yyyy aa bb [printer]	Hunt memory for aa bb (etc.)	4-3
H xxxx yyyy "text" [printer]	Hunt memory for text	4-3
L "filename" [n] [xxxx]	Load file into memory at xxxx	4-3
M xxxx	Display one line of memory	4-3
M xxxx yyyy [printer]	Display memory from xxxx to yyyy	4-3
R	Display registers	4-3
S "filename" [n] xxxx yyyy [zzzz]	Save program file from xxxx to yyyy, with zzzz offset load address	4-3
T xxxx yyyy zzzz	Transfer block from xxxx to yyyy to block at zzzz	4-4
V "filename" [n] [xxxx]	Verify file	4-4
W xxxx yyyy zzzz	Walk code	4-5
X	Exit to BASIC (protect file)	4-6
SHIFT X	Exit to BASIC (don't protect)	4-6
/	Display disk status	4-6
>[S0:FILE* etc.]	Disk commands	4-6
$[0:FILE* etc.]	Disk directory	4-6
:xxxx aa bb cc dd [etc.]	Write data to address	4-6
.xxxx aa bb cc [etc.]	Write data to address and disassemble	4-6

Assembler Directives

		Page ref.
.BYT	Define Bytes(s)	3-6
.WOR	Define Word(s)	3-7
.DBY	Define Word(s) in high, low order	3-7
LABEL =VALUE (or EXPRESSION)	Equate value	3-7
*** = address**	Set program counter	3-7
***=* + constant**	Allocate memory	3-8
.MEM source code **.NOMEM** source code	Assemble to memory	3-8
.LIST source code **.NOLIST** source code	List assembly	3-8
@ =address	Disk load address	3-9
.FIL **.END**	Linked Files (Disk or Cassette)	3-9
.LIB filename	Library files (Disk only)	3-9
.IF expression source code **[.ELSE]** source code **.ENDIF**	Conditional assembly	3-10
.END	End assembly	3-11

Text Editing Keys

Page 2-4

CRSR keys	Move the cursor in the indicated direction
RUN	Exit to the machine code monitor
STOP	Cancel line entry/command
CLR	Erase to the right of the cursor on the current line
HOME	Move current line to the centre of the screen
INST	Insert a space at the cursor
DEL	Delete character to the left of the cursor
RETURN	Next line/field
SHIFT RETURN	Next Find/Replace
CTRL	Pause listing

Editor Function Keys

Page 2-4

F1	Go to the end of the file
F2	Go to the start of the file
F3	Next screen
F4	Previous screen
F5	Place the cursor at the start of the next field
F6	Place the cursor at the end of the current line
F7	Insert blank line(s) above the cursor
F8	Highlight and delete line(s)

Editor Commands

Command	Description	Page ref.
C= A	Assemble the Source File	2-6
C= C	Copy Source Line(s)	2-8
C= D	Disk Commands	2-8
C= F	Find Sequence of Characters	2-9
C= G	Go To Label	2-10
C= I	Set Input/Output Device	2-10
C= L	Load File	2-11
C= M	Move Source Line(s)	2-11
C= N	New File - Clear File & Symbol Table Memory	2-12
C= O	Old File - Restore File Erased by New File Command	2-12
C= P	List File to Screen/Printer	2-12
C= Q	Query - Programmer's Calculator	2-13
C= R	Find & Replace Sequence of Characters	2-14
C= S	Save File	2-14
C= X	Mark Position	2-15
C= Y	Go To Marked Position	2-15
C= Z	Append File from Disk or Cassette	2-16
C= 1	Set Foreground Colour	2-16
C= 2	Set Background Colour	2-16
C= 3	Set Border Colour	2-17
C= 4	Set Default Screen Colours	2-17

Pagina Bianca

Ready64.org (2024)