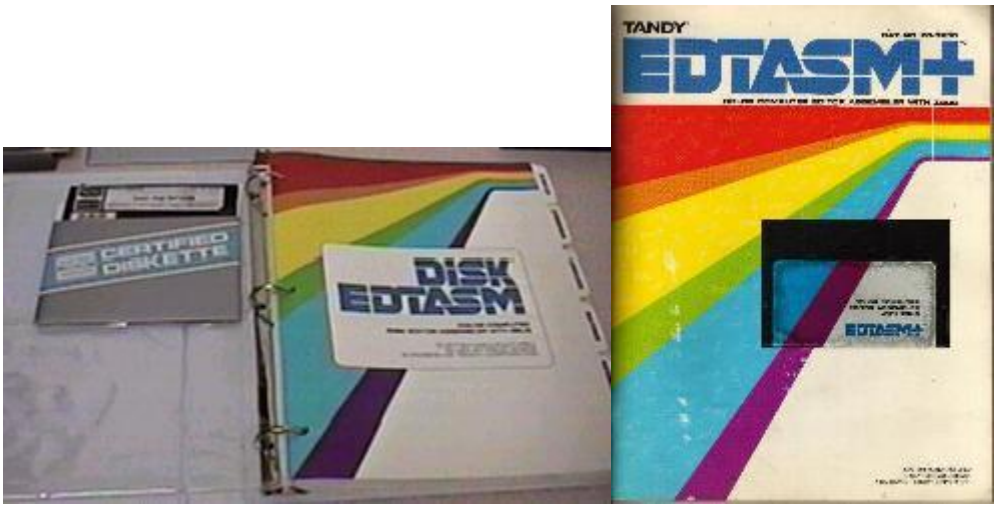
Edtasm Crash Course



Basic on the CoCo is a powerful language, but lack of memory space and slow execution times can be a problem.
If I want to crank out a program in an afternoon, I write it in Basic. If I have a Basic program that has some slow spots in it, I will usually write an assembly routine and replace that portion of Basic code. Any serious work though, I do strictly in Assembly Language. The fastest most memory efficient programs are written fully in assembler. The assembler I have some experience with is Edtasm.

If you have the original documentation, what follows may seem brief. If you have no documents, this may be enough to get you going. Edtasm may seem intimidating at first, especially if you have never used Assembly Language before. When I started, I found writing small routines that modified video memory on the text screen provided a starting point to learn Edtasm and the basics of assembly language programming. What follows is not a course in 6809 assembly, but more on how to operate the Edtasm program itself.

1. Starting it up.

Cartridge version. Plug in the cartridge, then turn the CoCo on. Program will start automatically, in the Editor/Assembler mode.

Disk version. With the computer and drive on, and the Edtasm disk loaded, type RUN"DOS" at the Basic prompt. This loads in a cheesy version of DOS. A menu will appear, select the item to execute a program. Type in the program name, EDTASM, and it will then load and automatically start.

2. The Editor/Assembler

At startup, you will have an asterix (*) for a prompt. This lets you know you are in the Editor. To start inputting lines of code, type;

I <enter> This is the insert command. The editor will automatically pop up a line number, 00100 to start.You can now enter your line of code, which may consist of up to 4 fields.You can see the start of each field by using the right arrow key to tab between the fields. Alternatively use the left arrow key to tab backwards.

The first field is for the symbol. Every line of code does not require a symbol. The symbol can be up to 6 characters long. Use the right arrow key to tab over to the second field done once your symbol (if any) is named. The second field is always required, this is your command. The third field is for the operand, the fourth for your comments if any. Once your line of code is complete, press <enter> and the Editor will advance to the next line ,00110.

Once your lines are complete, stop the Insert mode by pressing <break>. You will then be at the * prompt. If you are like me, you will want to edit your listing. Edtasm's editor is a lot more useful than the editor in Extended Basic. Here is a list of commands;

A Assemble, see Lesson 3 for a description.

C Copy Copies a range of lines to a new location. Must include startline, range and increment. For example;

C600, 100:200,10 Copies existing lines between 100 and 200 to a new location starting at 600 with an increment of 10.

D Delete Deletes a line or range of lines. For example;

D100 Deletes line 100. **D 100:200** Deletes all lines from 100 to 200

E Edit Edits a line. For example;

E100 Opens line 100 for editing.

F Find Finds a string of characters. For example;

Fword will search for the string word.

H Hardcopy Outputs to the printer, can be a single line, a range or the entire listing. For example;

H100 Prints out line 100, **H100:200** prints out lines 100 to 200, **H#:*** prints the entire listing.

I Insert Inserts lines. You can specify a startline and an increment. For example;

I starts inserting at default 100 or past the current line number with a default increment of 10.

I600,100 Inserts starting at line 600 with an increment of 100.

L Loads from cassette. Loads the first available file, unless a name is specified. For example;

L FILE Will load in a source file with the name FILE.

LD Loads from disk. Of course this only works if you are using Disk Edtasm

LDA Loads from disk and appends to the current edit buffer's contents.

M Move command. Same as copy except the original lines are deleted.

N Renumber Can specify a startline and the increment. For example;
N100,2 Renumbers starting at line 100 with an increment of 2.

O Displays available memory, Disk Edtasm only.

P Prints a line or range of lines on the screen. Similiar to Basic's LIST command.
P100 Displays line 100, **P100:200** Displays lines 100 to 200. **P#:*** Displays the entire listing

Q Exits to Basic

R Replace. Replaces startline, can specify an increment also

T Same as **H** command, except no line numbers are printed. I have no idea why anyone would want to do this.

V Verify Checks your cassette tape file for errors. Can specify a filename, for example;
VABC Checks the file named ABC for errors.

W Writes to cassette. Can specify a filename

WD Writes to disk. Can specify a filename

Z Starts the ZBUG debugger

3. The Assembler

This is the easy part, the computer does all the work for a change. You will appreciate this if you have ever done any hand coding. Once your source program has been edited to your satisfaction, start the assembler from the * prompt (Editor). The most common assembler command I use is **A/IM/WE**

The switches;

/AO Absolute origin Use this to force the assembler to locate the object code in a specific place.

/IM In memory Object code is assembled into memory

/LP Line printer Listing is printed out

/MO Manual origin Another method to locate the object code

/NL No listing No listing to screen

/NO No object No object code output

/NS No symbol table No symbol table listed

/SR Single record Never used this one, don't know what it is.

/SS Short screen listing Makes the listing slightly different

/WE Wait on errors Stops the assembly at any line with an error.

/WS With symbols Self explanatory

If you have a short program to assemble, the **A/IM/WE** will work fine. If your program gets large, watching the listing go by on the screen seems to take forever. I usually use **A/IM/WE/NL/NS** to speed up the assembly. You are looking for 0000 errors at the end of listing. If your program has a lot of symbols, and the **/WE** switch is not on, you may not see the error count, so either turn on the **/NS switch** or include the **/WE** switch. Starting a program that has errors in it is a sure way to crash your computer. If you have errors, go back, edit the listing, and reassemble. Edtasm helps you here as it shows the lines with the error and tells you the type of error.

4. The debugger ZBUG

You have an error free listing, it is safe (maybe) to run the program. From the Editor's * prompt, type **Z** to enter ZBUG.
The program will switch to a # prompt.

The most common command is **G,** the Go command. Type in **G** and the name of the symbol where your program starts.
I always label my programs start as **START,** so my start command is always **GSTART.**

Likely your program will need some work, so this is where the debugger comes in. See the following list of commands.

A Set examination mode to ASCII output

B Set examination mode to Byte output

C Continue Restarts execution of your program after a breakpoint has stopped it.

D Display Shows all breakpoints that have been set.

E Editor Exits Zbug and returns you to the Editor

G Go The program is started at this program line
GSTART will start execution at with the line with START for a symbol.

H Set display mode to half symbolic

I Set the input base. Can be 8, 10 (decimal), 16 (hexadecimal)

L (Cartridge Edtasm) Load Loads machine code from tape. Can specify a filename

LC (Disk Edtasm) Load Loads machine code from tape. Can specify a filename

LD (Disk Edtasm) Load Loads machine code from disk. Can specify a name and a
loading offset if required.

LDS (Disk Edtasm) Load Loads machine code and the appended symbol table. Can
specify filename, and load offsets for the code and the symbol table.

N Set display mode to Numeric

O Set the output base. Can be 8, 10 (decimal), 16 (hexadecimal)

P (Cartridge Edtasm) Save Saves machine code to tape. Specify start address, end address,
and execution start address

PC (Disk Edtasm) Save Saves machine code to tape. Specify start address, end address,
and execution start address

PD (Disk Edtasm) Save Saves machine code or memory to disk. Specify start address,
end address, and execution start address

PDS (Disk Edtasm) Save Saves machine code to disk and appends the symbol table
Specify start address, end address, and execution address.

R Registers Displays the contents of all registers. You usually use this right after a breakpoint
when debugging.

S Set display mode to Symbolic

T Display Shows contents of memory on screen. Must specify a start and end address.

TH Print memory to printer. Must specify a start and end address.

U Copies memory. Must specify a source address, destination address, number of bytes

X Set breakpoint. Must specify an address.(hard way) Just specify a symbol(easy way)

Y Yank breakpoint. If you specify an address it removes that breakpoint. By itself it removes all
breakpoints.

The Zbug Calculator

You will notice that Zbug displays its findings in hex. You can change this by typing **I** or **O** to change the input or display modes. A **I10** command will set the input to base 10 decimal. If you want to set it back to hex, **I16** will do it. To get the calculator to perform a decimal to hex conversion, type;

I10
O16
65535=

In this example I want to see what 65535 is in hex, Zbug will output FFFF.

To perform a hex to decimal conversion, the command is;

I16
O10

The calculator has a fair range of available operators

+ Addition or positive

- Subtraction or negative

***** Multiplication

< Shift

.AND. Logical AND

.DIV. Division

.EQU. Equals

.MOD. Modulus

.NEQ. Not equal

.NOT. Complement

.OR. Logical OR

.XOR. Exclusive OR

The best thing about Zbug's calculator is it will operate with not only numbers in any base, but also with symbols and ASCII characters. It only operates on whole numbers (integers), so don't try to do any math that will have anything to the right of the decimal point.

Out of time for now, will load more here ASAP. For now see the following sample;

Start your editor and enter the following

```
*I
00100 START   JSR     43304       CLS SUBROUTINE
00110             LDA     #65           ASCII A
00120 AGAIN   JSR     [40962]   CHROUT SUBROUTINE
```

```
00130                   INCA                    NEXT IN ALPHABET
00140                   CMPA   #91          ONE PAST ASCII Z?
00150                   BNE      AGAIN
00160                   LDA      #32          ALPHABET DONE
00170                   LDB      #6           PRINT 6 SPACES
00180 REPEAT JSR        [40962]
00190                   DECB
00200                   BNE     REPEAT
00210 LOOP      BRA     LOOP
00210                   END
```

<break>
*A/IM/WE
program assembles, must be 0 errors!
*Z
X LOOP
GSTART

Your screen will clear, the alphabet will print at the top of the screen, and then Edtasm will break the program and return you to ZBUG.

Edtasm Patches

80 Column mode for Disk Edtasm+ with a CoCo3

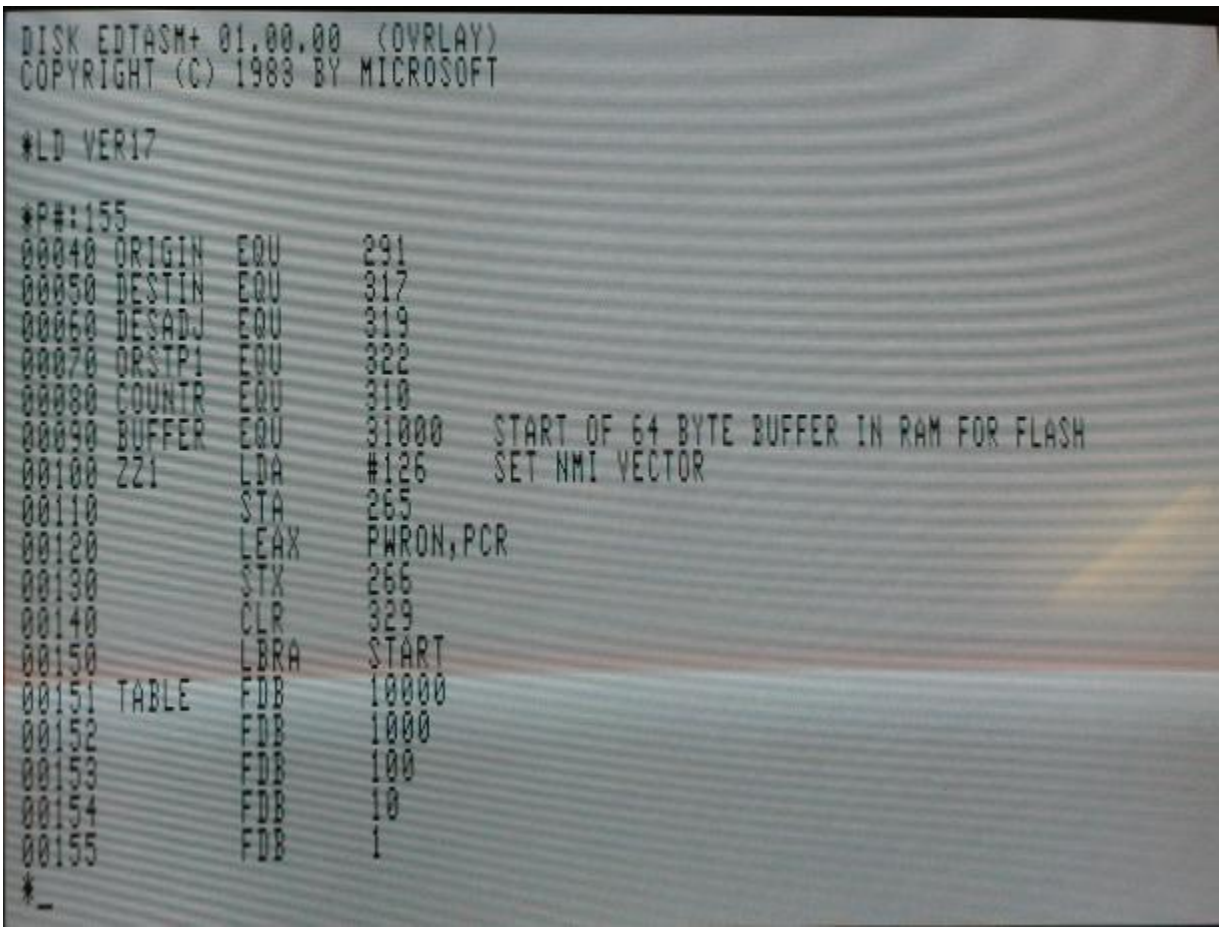
This patch will autostart the Disk Edtasm program in a 80 column mode.

Load in the DOS/BAS program from the Edtasm disk. Add these two lines;

0 WIDTH80:PALETTE 0,63:ATTR 0,0:CLS1
2 REMEDTASM

Save the DOS/BAS program back to disk. Be sure you do this to your working copy, not your original disk.

The next time you run DOS/BAS, it will be 80 columns wide, black characters on a white screen. If you are running out of buffer space, edit line 2 to load EDTASMOV instead of EDTASM, gives a little extra room for editing. See my fuzzy screenshot below for a typical screen in this mode.



Last update May 18/2002

Index