

# Disassembler Window Tutorial

---

Author: E J Jaquay

This document was prepared to give the reader a quick overview of the new VCC Debugger Disassembler and how it might be used to study Color Computer machine language programs. This tutorial demonstrates features available with VCC version 2.1.9.1 and thereafter.

The best way to use this tutorial is to fire up VCC and actually do the steps outlined in this tutorial but first some background.

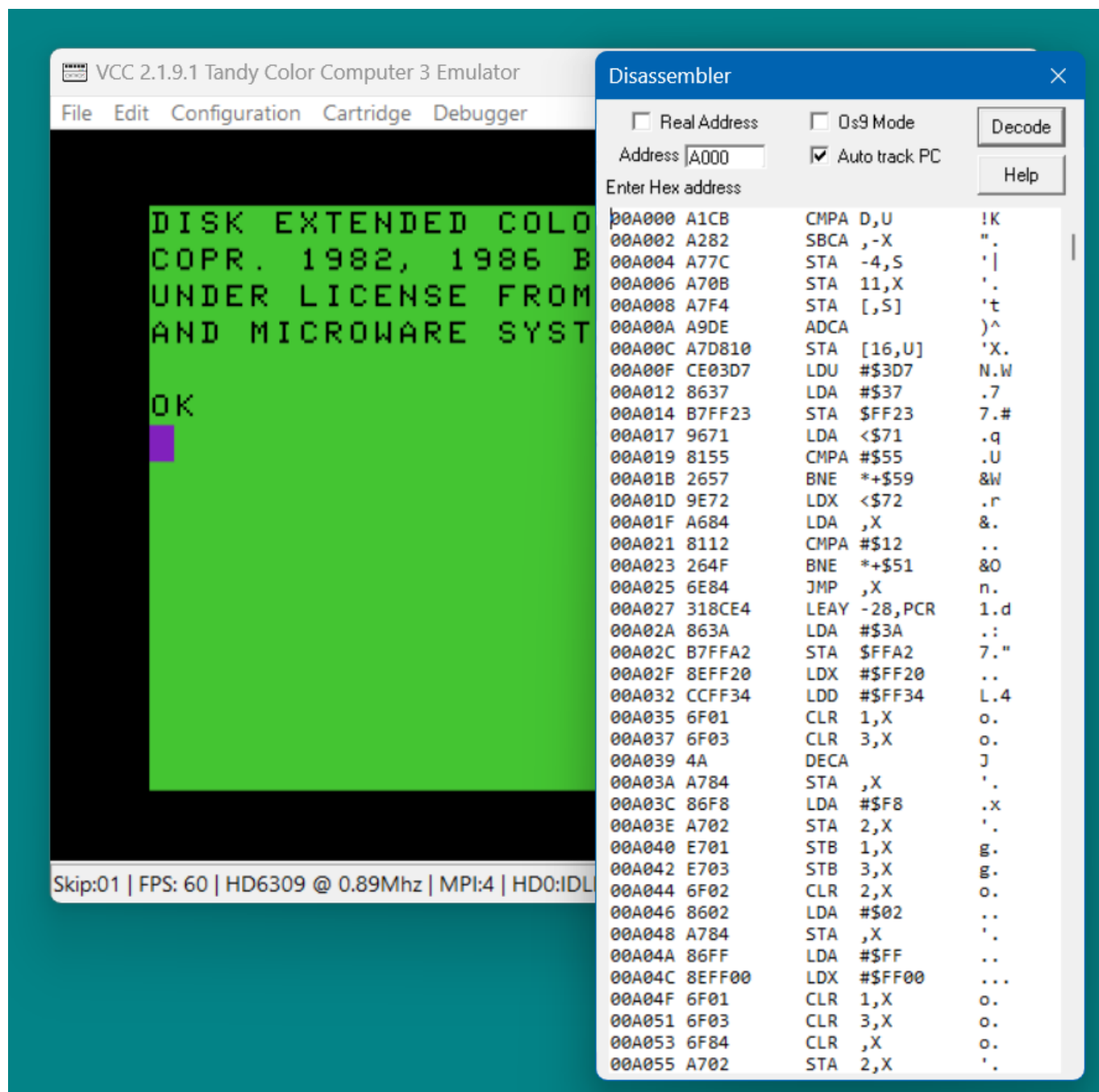
The Disassembler Window started as way to use opcode decoder logic that was added to VCC for the Debugger Execution Trace feature. It was fairly simple to use the decode logic for the disassembler. It worked reasonably well for DECB programs but not so well for looking at Nitros9 code. This was because Nitros9 is bank switching memory blocks in and out of CPU space frequently while DECB for the most part does not.

The disassembler was improved to allow users to use either CPU (0-FFFF) or real (0-1FFFFFF) addressing. This allowed users to view Nitros9 code anywhere in physical memory by either entering the absolute address or the block and offset provided by Nitros9's mdir command. This works well except in the case where modules cross block boundaries that are not actually contiguous in real memory.

Once we could disassemble code we wanted to be able to use the decoded text for debugging. So the debugger was enhanced to allow setting of breakpoints to track the CPU program counter when paused.

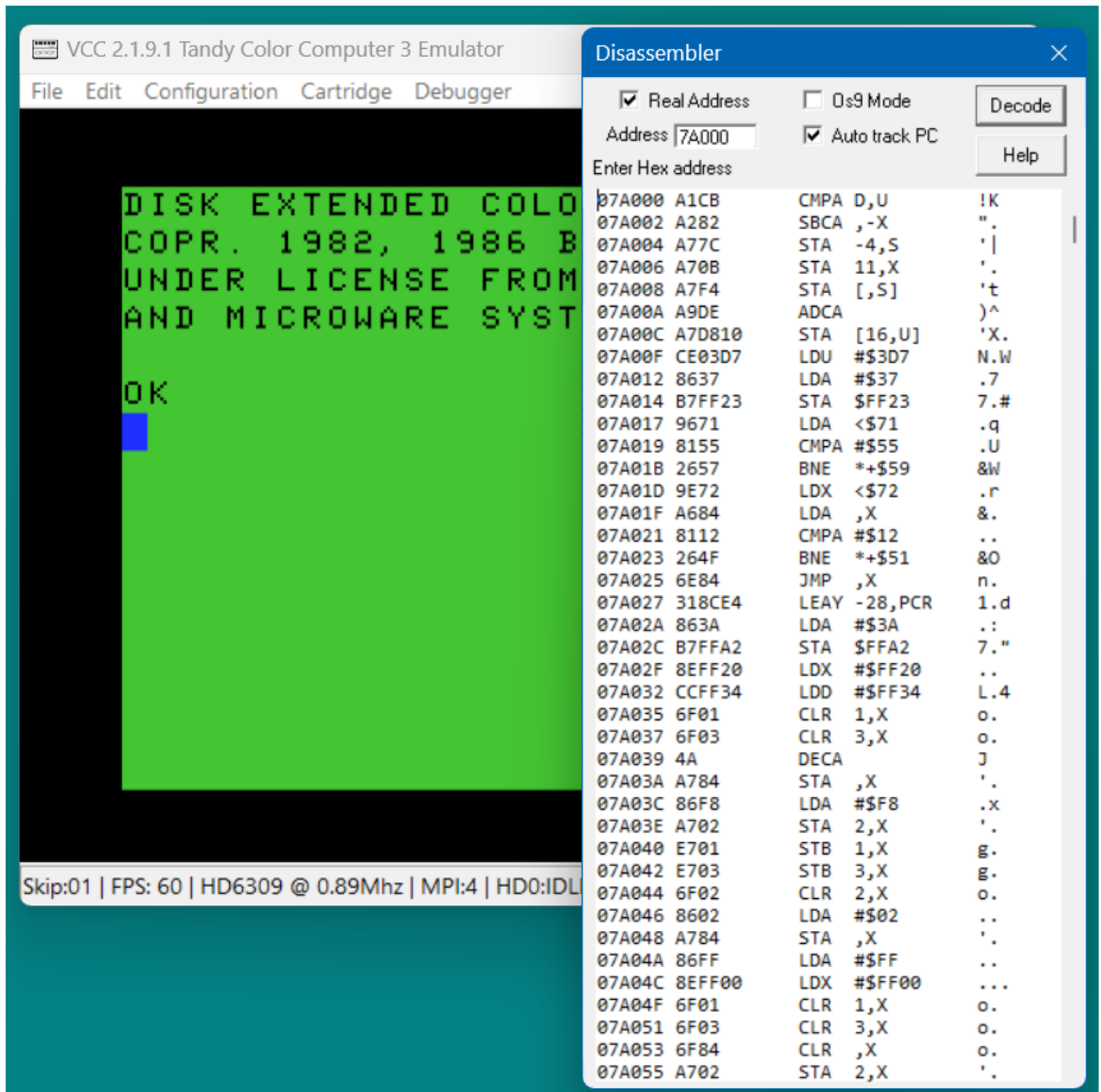
## Basic disassembly

Select "Debugger" then "Disassembler." The disassembler window comes up. In the "Address" field enter 'A000' (All addresses are entered in hexadecimal) then press the keyboard Enter key or click on the Decode button. Disassembly text starting from address A000 is shown. (Vcc should be configured to have at least 512K of memory and DECB running to get this result)



## Real vs CPU addressing

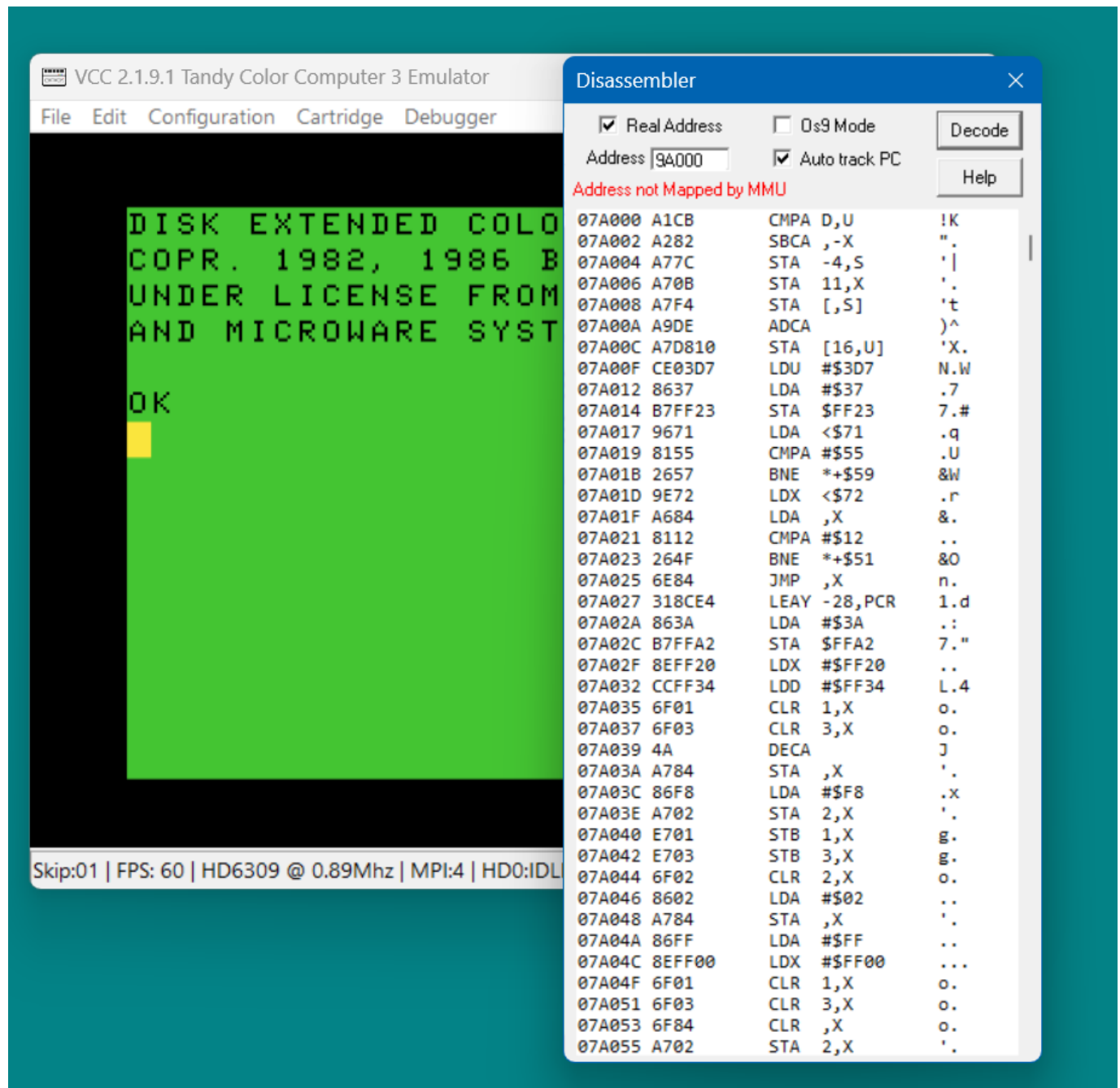
The disassembly above is shown using the CPU's view of memory. Sometimes it is necessary to view real memory. Real memory consists of many 8K blocks. The MMU can map only eight of them at a time to CPU memory but the disassembler is capable of decoding programs in all of real memory. To see real memory press the 'M' key. This key toggles the disassembly address between CPU and real.



The disassembly is changed to show real addresses and the "Real Mem" box is now checked. Note that simply checking or unchecking the "Real Mem" box does not change the disassembly.

You must use the "M" key to remap.

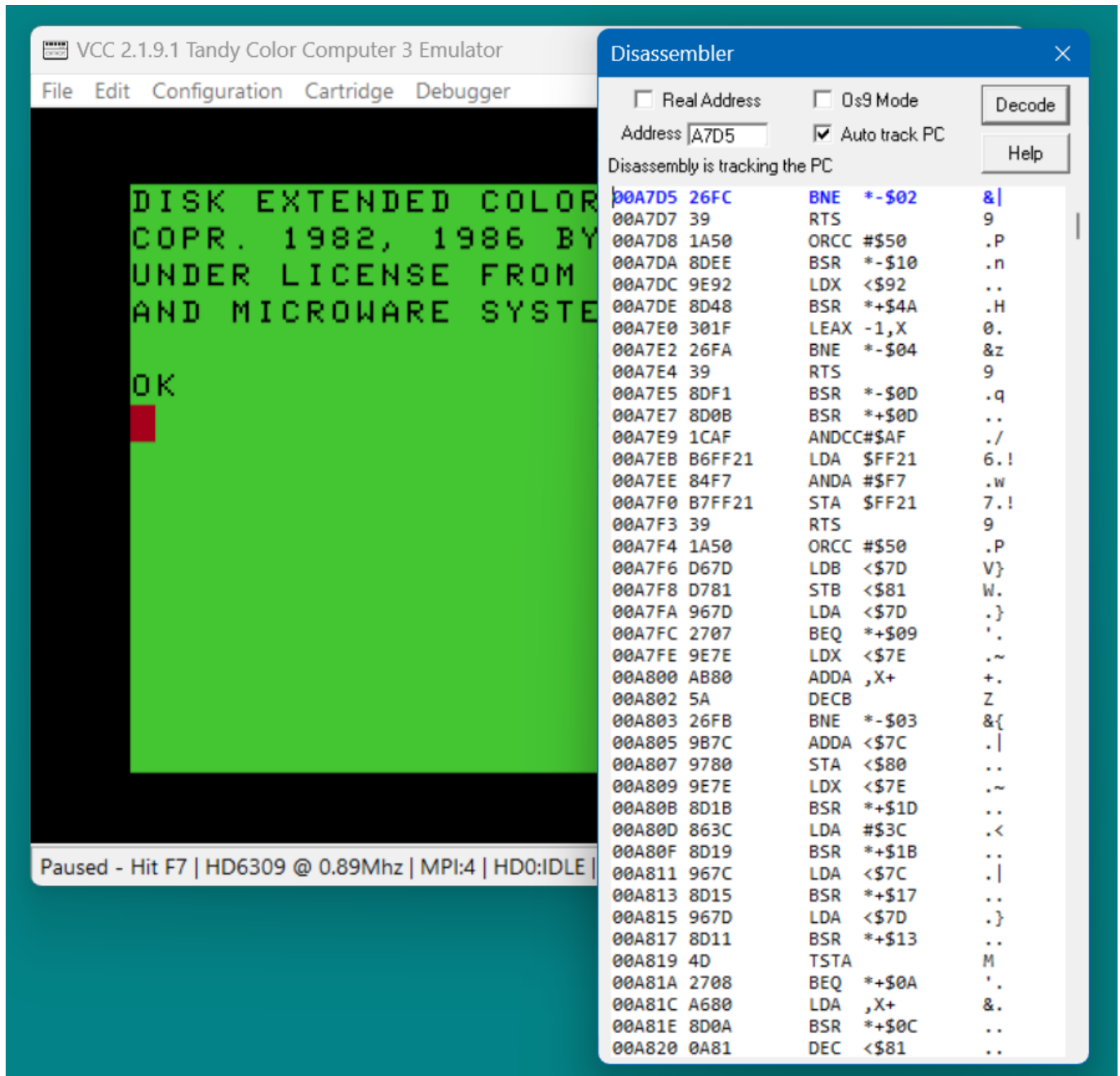
You can toggle between real and CPU addressing at will by pressing the "M" key however it will not always work. If the memory displayed is not currently mapped to CPU memory remapping will fail with an error message.



Here while using real addressing I changed the address from 7A000 to 9A000 and tried "M" to remap and got an error. When I changed the address back to 7A000 the "M" toggle worked again because that block is mapped to the CPU.

## Pausing and viewing code.

By default when the cpu is paused the Disassembler will switch to CPU addressing mode and then find and highlight the code at the current PC. This lets you see what the software was doing. This behaviour can be changed by unchecking the "Auto Track" checkbox. The Vcc "F7" key, "Pause" from the "File" menu, or the Disassembler 'P' key can be used to pause the CPU.

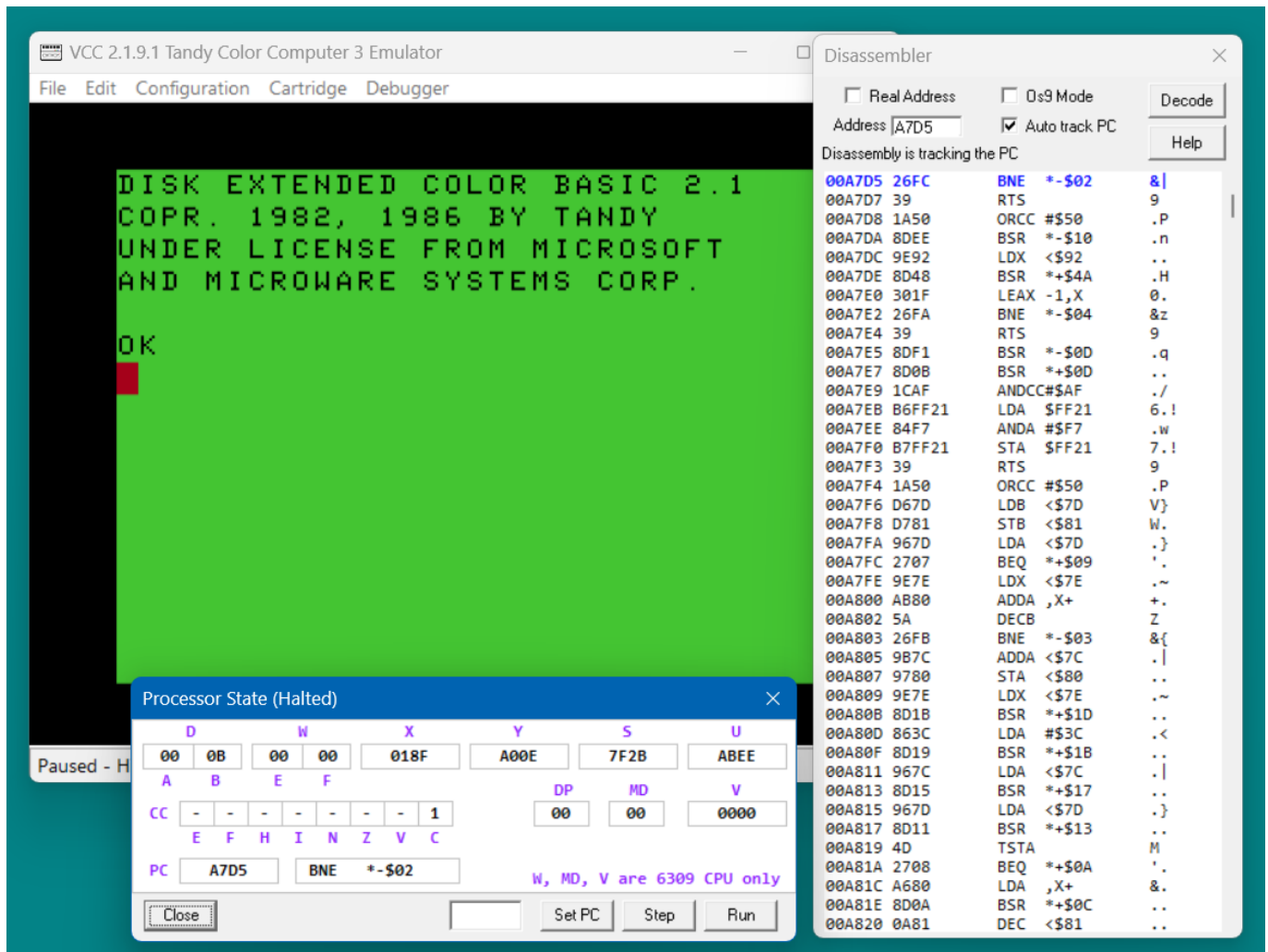


You can view the code that was executing at anytime by starting the Disassembler and pressing the 'P' key. The screenshot above shows the result if this is done while DECB is at the color prompt. In the Disassembler the address starting at A7D5 has automatically been decoded. The blue highlight indicates the program counter address. Pressing 'G' will unpause the CPU. If you

toggle pause several times you will notice the highlight goes away when the CPU is running and sometimes the PC pauses at A7D3 instead of A7D5.

## The Processor State Window

The "Processor State" window shows the current CPU register values. If you press "I" from the disassembler window the processor state window will be launched without using VCC's Debugger menu.

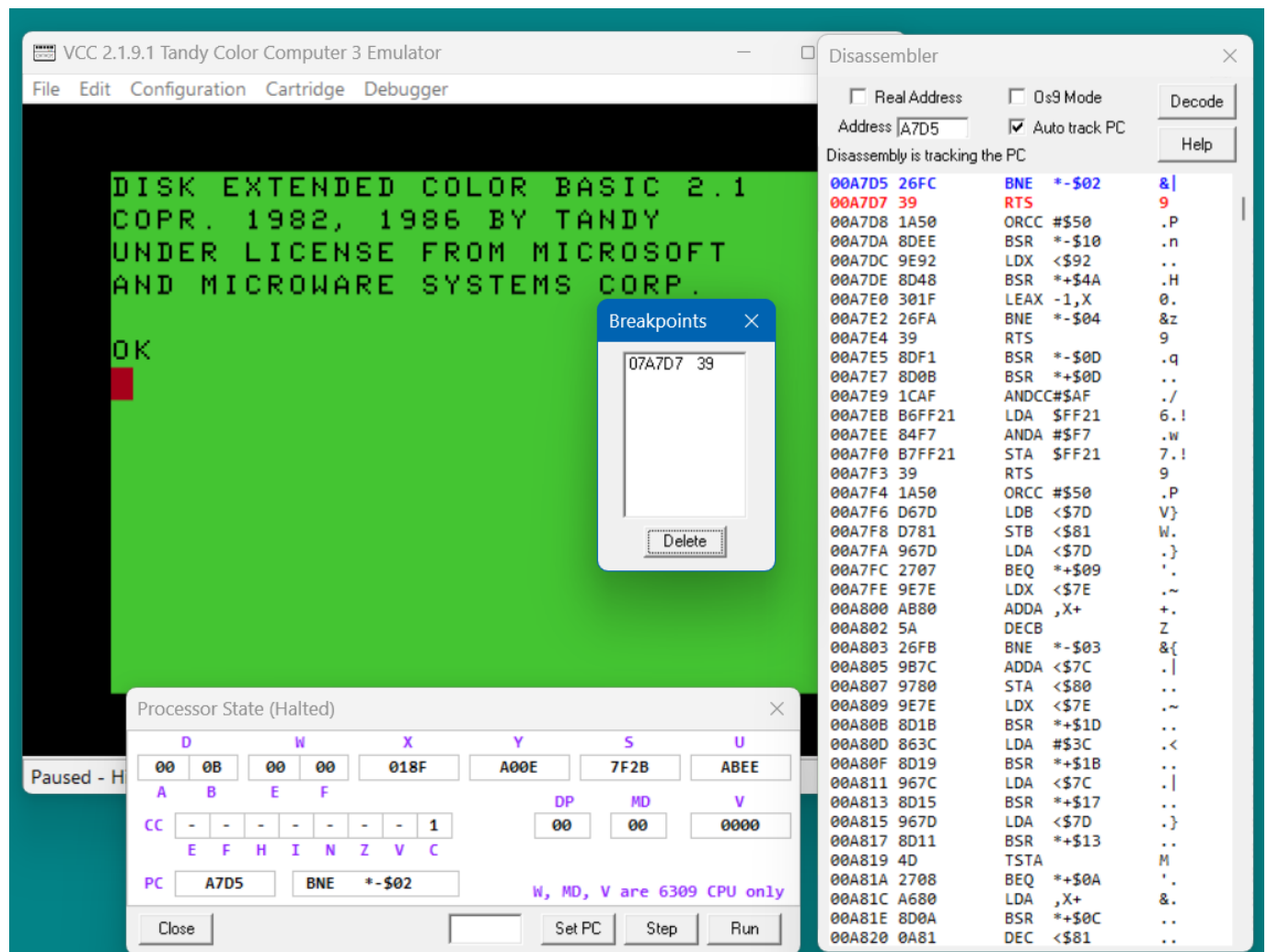


You can use Processor State "Step" button or the Disassembler 'S' key to step through code. If you keep stepping you can see that the CPU is in a loop that changes the colors of the DECB cursor. Most of the time the CPU running a timing subroutine that counts the X register down to zero.



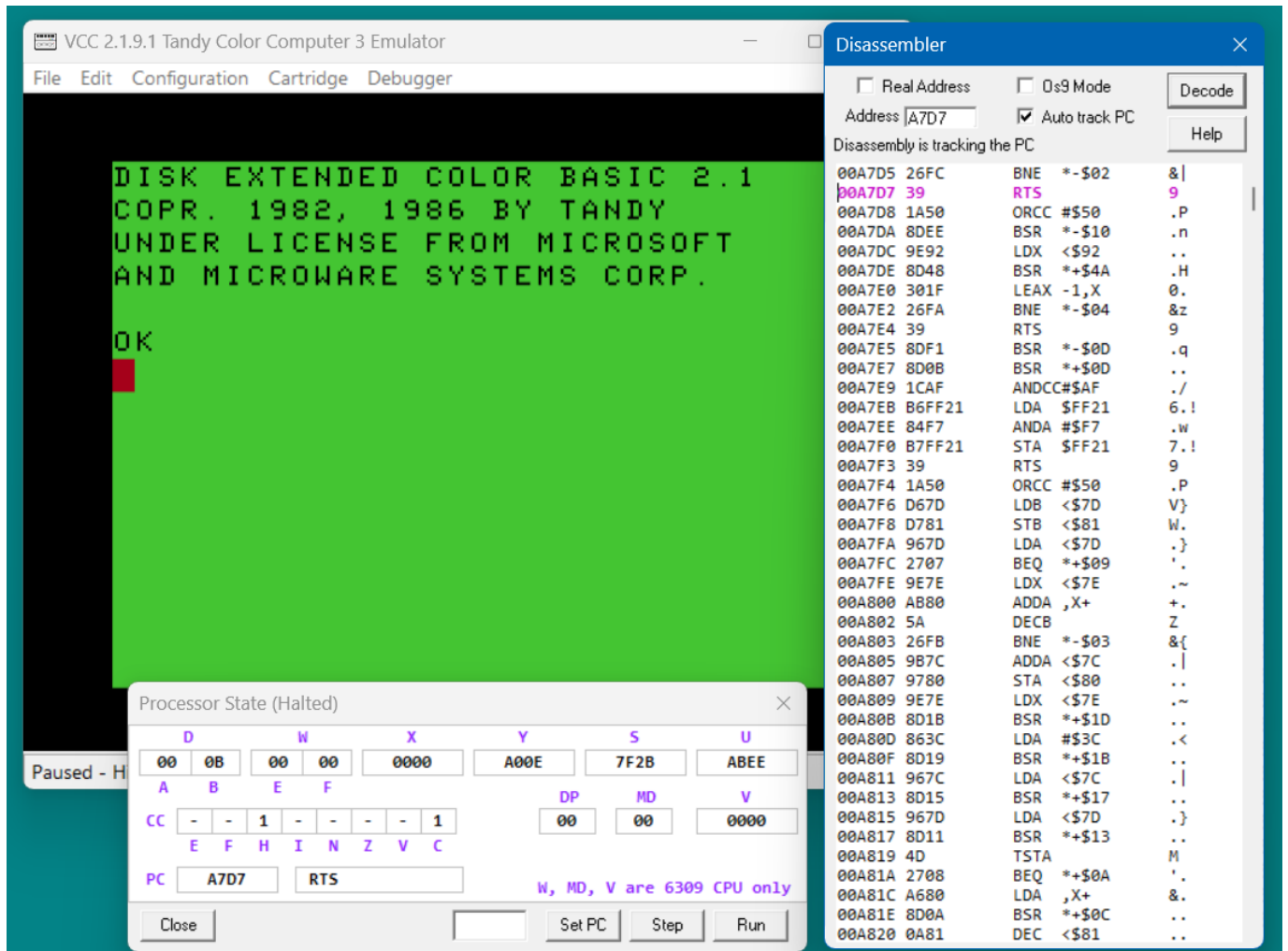
## Setting a breakpoint

With Vcc halted at the DECB prompt set a breakpoint on the RTS instruction at A7D7. Decode and use the mouse to click on the line containing the RTS and press the 'B' key. The address at that line will turn red to indicate that there is a breakpoint there. If you press the 'L' key a listbox will show all the break points that have been set. (Don't worry about remembering all these keys, clicking on the HELP button will show them)



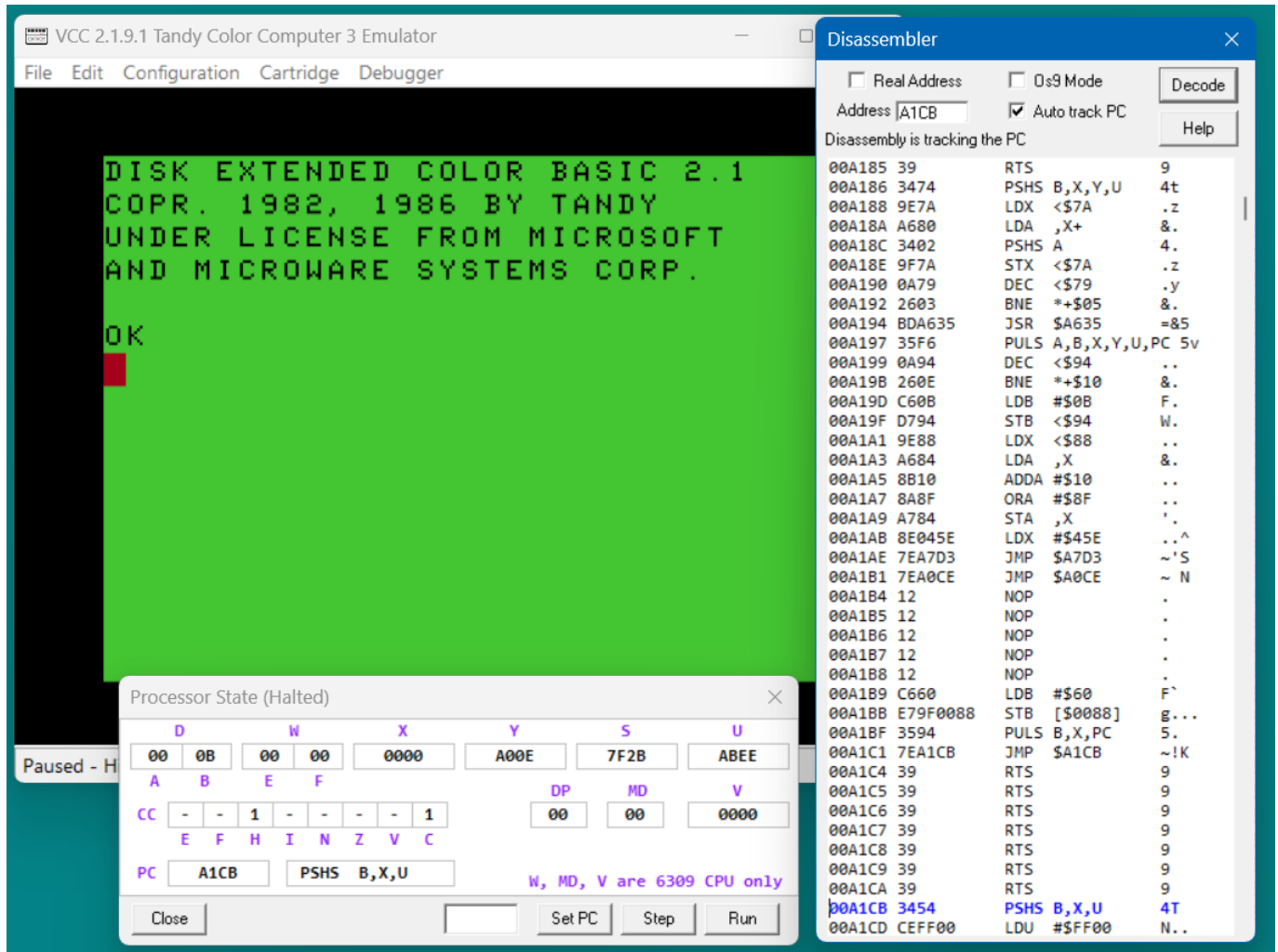
The "Breakpoints" listbox shows the breakpoint's real address and the original opcode. A breakpoint can be removed by selecting it and pressing "Delete".

With the breakpoint set at address A7D7 use the "G" or Run button to start the CPU. The CPU immediately pauses and the RTS instruction highlight has changed color from red to magenta. It is that color because the PC is at the breakpoint. Notice that pressing "G" again and again seems to do nothing but if hold the hold "G" down to auto repeat you should see the DECB cursor occasionally changing color. What is happening is the CPU is stopping everytime it encounters the breakpoint at the RTS.



You can step however. 'S' will step one instruction which will execute the RTS to return to the caller and stop.

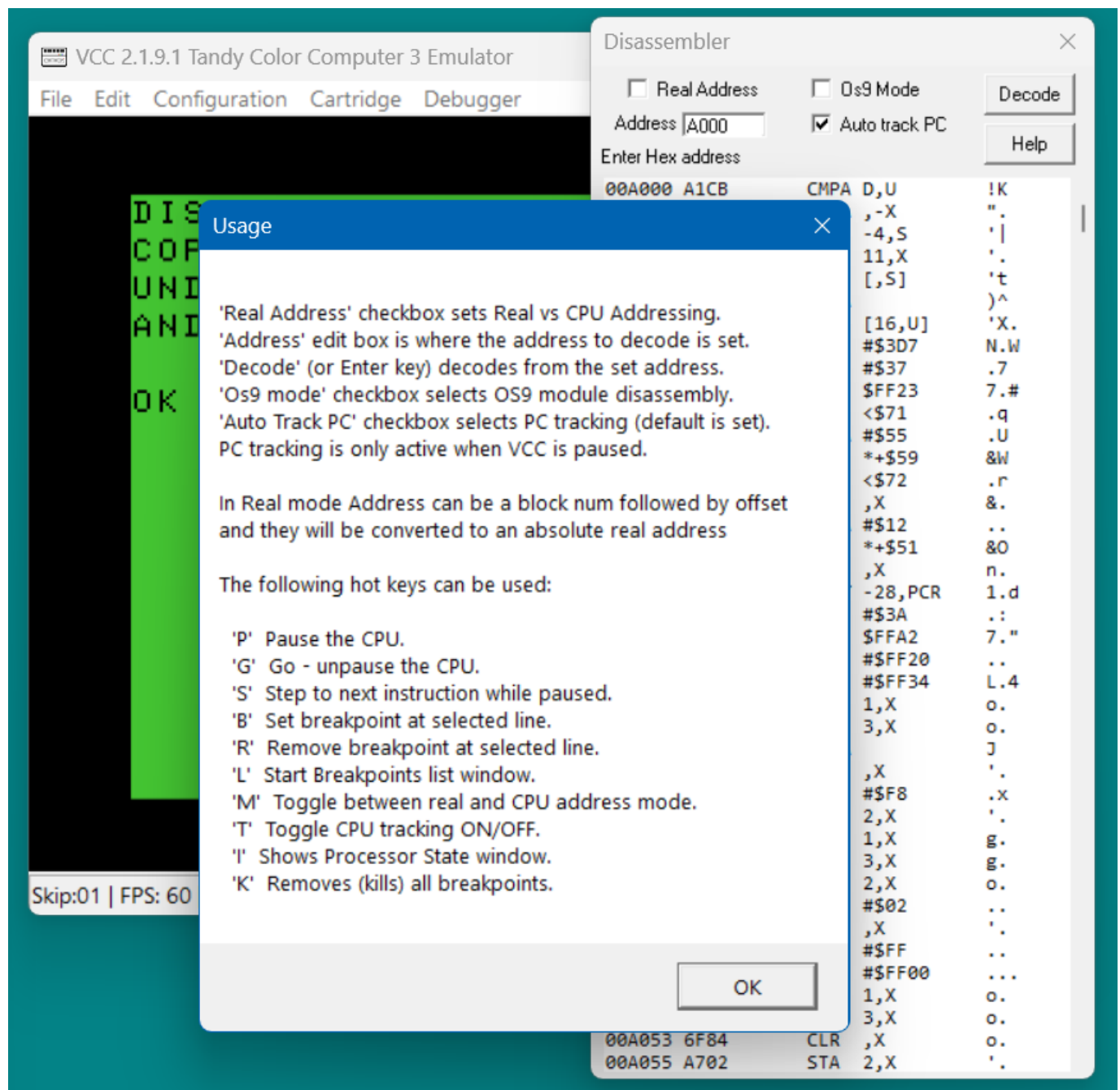
The next instruction is another subroutine call. Stepping again brings the PC to the first instruction in that subroutine. The PC is now at the bottom of the disassembly. This is because the disassembler found the already decoded address and scrolled the text to make the line visible. You can continue to step through the code and the disassembler will continue to follow the program counter.



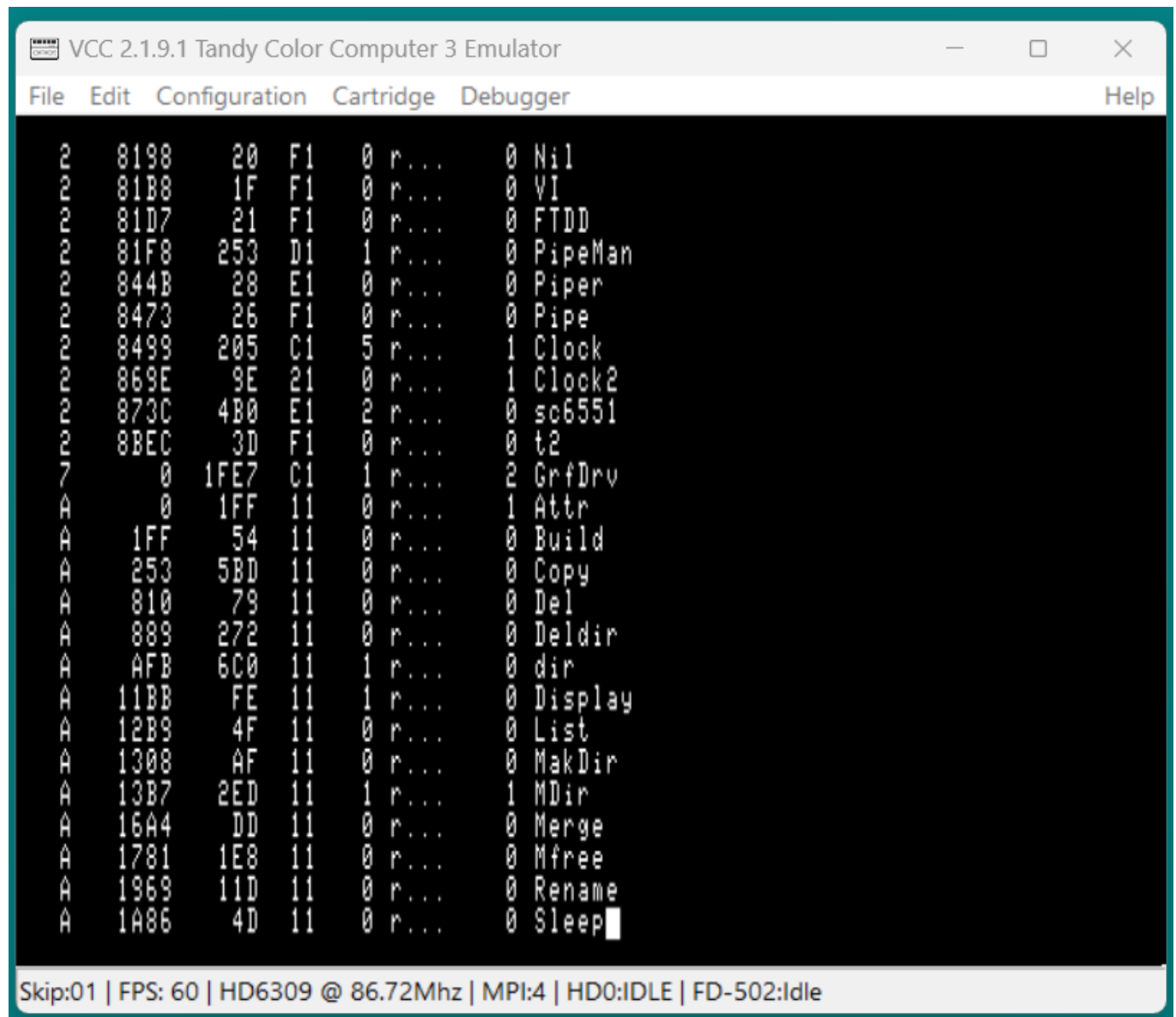
A breakpoint can be removed by selecting it in the disassembly screen and pressing 'R' key. Or use the breakpoints listbox to select and delete it. Also you can press the 'K' key to remove all breakpoints. Exiting the Disassembly screen will also remove them.

## Help Message Box

The "Help" button brings up a brief help message.



## Working with Nitros9

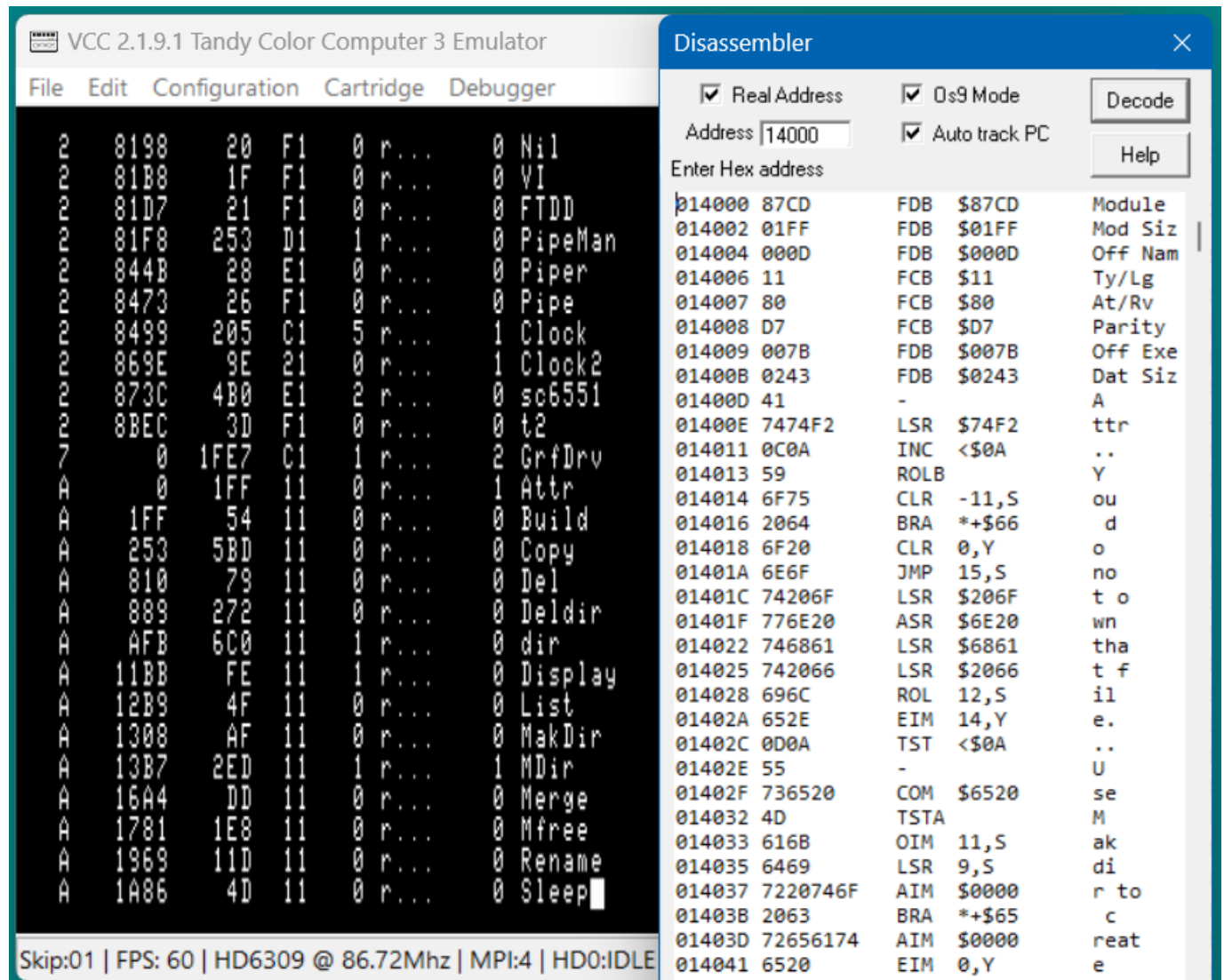


The screenshot shows the VCC 2.1.9.1 Tandy Color Computer 3 Emulator window. The menu bar includes File, Edit, Configuration, Cartridge, Debugger, and Help. The main display area shows a memory dump with columns for address, offset, hex value, and module name. The dump lists modules like Nil, VI, FTDD, PipeMan, Piper, Pipe, Clock, Clock2, sc6551, t2, GrfDrv, Attr, Build, Copy, Del, Deldir, dir, Display, List, MakDir, MDir, Merge, Mfree, Rename, and Sleep. The status bar at the bottom displays: Skip:01 | FPS: 60 | HD6309 @ 86.72Mhz | MPI:4 | HD0:IDLE | FD-502:Idle.

Address	Offset	Hex Value	Module Name
2 8198	20	F1	0 r... 0 Nil
2 81B8	1F	F1	0 r... 0 VI
2 81D7	21	F1	0 r... 0 FTDD
2 81F8	253	D1	1 r... 0 PipeMan
2 844B	28	E1	0 r... 0 Piper
2 8473	26	F1	0 r... 0 Pipe
2 8499	205	C1	5 r... 1 Clock
2 869E	9E	21	0 r... 1 Clock2
2 873C	4B0	E1	2 r... 0 sc6551
2 8BEC	3D	F1	0 r... 0 t2
7 0	1FE7	C1	1 r... 2 GrfDrv
A 0	1FF	11	0 r... 1 Attr
A 1FF	54	11	0 r... 0 Build
A 253	5BD	11	0 r... 0 Copy
A 810	79	11	0 r... 0 Del
A 889	272	11	0 r... 0 Deldir
A AFB	6C0	11	1 r... 0 dir
A 11BB	FE	11	1 r... 0 Display
A 12B9	4F	11	0 r... 0 List
A 1308	AF	11	0 r... 0 MakDir
A 13B7	2ED	11	1 r... 1 MDir
A 16A4	DD	11	0 r... 0 Merge
A 1781	1E8	11	0 r... 0 Mfree
A 1969	11D	11	0 r... 0 Rename
A 1A86	4D	11	0 r... 0 Sleep

Working with Nitros9 can be difficult because it loads modules in real memory then maps them as needed. In this example we are working with the "Attr" module. Here "mdir -e" shows that Attr is located in block "A" with an offset of "0". It is possible that Attr will be loaded elsewhere on your system but the steps are the same.

Next I started the Disassembler window and checked both the "Real Address" and "Os9 Mode" boxes. In real memory mode the disassembler allows you to enter addresses as a block number followed by an offset. It will automatically convert these to the absolute address. So to view the attr module I entered "A 0" in the Address field. Then I hit enter to get the disassembly. The disassembler automatically converted my "A 0" to "014000".



This shows one of the features of Os9 Mode, the Module header has been decoded. At address 01400D I can see that this is indeed the "Attr" module. The "Os9 Mode" checkbox enables module header decoding.



At 014009 I see the execution offset is 007B. I add this to the module start offset (cleverly I choose a module with an zero offset) and enter the block and offset 'A 7B', in the Address field.

The screenshot shows the VCC 2.1.9.1 Tandy Color Computer 3 Emulator interface. The main window displays a memory dump with columns for address, hex value, offset, and module name. The disassembler window is open on the right, showing the disassembly of the code starting at address 01407B.

**Memory Dump (Left Panel):**

Address	Hex	Offset	Module
2	8198	20	F1
2	81B8	1F	F1
2	81D7	21	F1
2	81F8	253	D1
2	844B	28	E1
2	8473	26	F1
2	8499	205	C1
2	869E	9E	21
2	873C	4B0	E1
2	8BEC	3D	F1
7	0	1FE7	C1
A	0	1FF	11
A	1FF	54	11
A	253	5BD	11
A	810	79	11
A	889	272	11
A	AFB	6C0	11
A	11BB	FE	11
A	12B9	4F	11
A	1308	AF	11
A	13B7	2ED	11
A	16A4	DD	11
A	1781	1E8	11
A	1969	11D	11
A	1A86	4D	11

**Disassembler (Right Panel):**

Address: 01407B

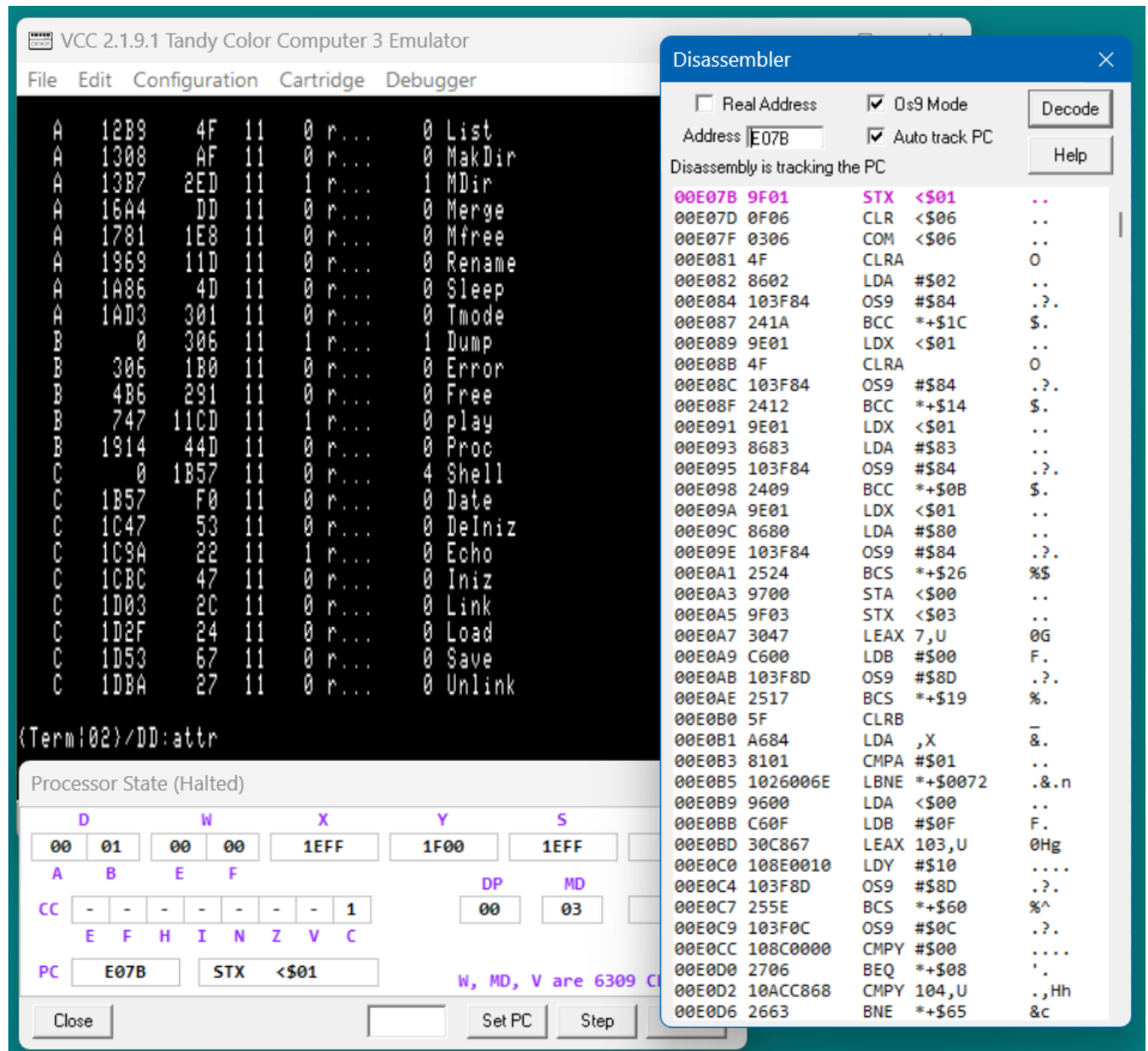
Enter Hex address: 01407B

Address	Hex	Offset	Module
01407B	9F01	STX	<\$01
01407D	0F06	CLR	<\$06
01407F	0306	COM	<\$06
014081	4F	CLRA	
014082	8602	LDA	#\$02
014084	103F84	OS9	#\$84
014087	241A	BCC	*+\$1C
014089	9E01	LDX	<\$01
01408B	4F	CLRA	
01408C	103F84	OS9	#\$84
01408F	2412	BCC	*+\$14
014091	9E01	LDX	<\$01
014093	8683	LDA	#\$83
014095	103F84	OS9	#\$84
014098	2409	BCC	*+\$0B
01409A	9E01	LDX	<\$01
01409C	8680	LDA	#\$80
01409E	103F84	OS9	#\$84
0140A1	2524	BCS	*+\$26
0140A3	9700	STA	<\$00
0140A5	9F03	STX	<\$03
0140A7	3047	LEAX	7,U
0140A9	C600	LDB	#\$00
0140AB	103F8D	OS9	#\$8D
0140AE	2517	BCS	*+\$19
0140B0	5F	CLRB	
0140B1	A684	LDA	,X
0140B3	8101	CMPA	#\$01
0140B5	1026006E	LBNE	*+\$0072
0140B9	9600	LDA	<\$00
0140BB	C60F	LDB	#\$0F
0140BD	30C867	LEAX	103,U
0140C0	108E0010	LDY	#\$10

At the bottom of the emulator window, the status bar shows: Skip:01 | FPS: 60 | HD6309 @ 86.72Mhz | MPI:4 | HD0:IDLE

The disassembly now starts at 01407B which is the start address of the Attr module. I set a breakpoint there.

From Nitros9 I typed in attr. NitroS9 mapped the page the attr module was in to CPU memory and ran it. The CPU halted at my breakpoint and the disassembly switched to CPU addressing. I can now step through the attr code and watch it work.



Here you can see another feature of Os9 Mode. SWI2 instructions have been converted to "OS9" instructions. 'OS9 #\$84' is a system call to open an existing file.



Enjoy!