5110VEMU

voidstar's Wintel Compatible IBM 5110 Emulator

Introduction

Before the IBM PC 5150 in 1981, before the Commodore, Apple, Tandy trinity of 1977, before the Altair 8800 kit of 1975 – there was the SCAMP prototype of 1973: Special Computer, APL Machine Portable. The prototype transitioned into a commercial product in 1975: the IBM 5100, with base of 16KB RWS/RAM, internal tape storage, and various external communication options.

A key feature in the development of the IBM 5100 was the technique of processor emulation. A core 16-bit processor was developed (called PALM, with 32x16-bit registers across four interrupt levels) that coordinated the execution of native System/370 (for APL) and System/3 (for BASIC) instructions, allowing re-use of many existing programs from those mainframes.

The IBM 5100 represents a key transition between the large mainframes of the 1960s, to the much more compact personal computers of the 1980s (the PDP-11 costing around $100,000, the IBM 5100 costing around $10,000, and desktop PCs around 1980 reaching $1,000). I think of all those college students and professionals that paved the way in the 1960s: to craft a compiler out of pure machine code, and mature the ideas and protocols of things like remote desktop and integrated debugging. These are the unsung heroes, to which the "kids" of 1977 owe their prosperity to.

Technical details about the PALM processor is difficult to come by, as it is based on proprietary IBM in-house work. Around 2007, Christian Corti performed two pioneer acts to help preserve this knowledge: (1) he used a still-functional 8" disk drive connected to an IBM 5110 to archive the Executable ROS, (2) he wrote a functional PALM emulator in C, based on notes he had collected and his own reverse engineering experience. These two things combined led to a functional emulation of the IBM 5110 system as a whole.

Christian Corti's original emulator used X11 GUI interfaces, which is typically available on Unix-like operation system (or under Windows with the Cygwin package). This included an excellent reproduction of the IBM screen font by Thomas A. Fine (of Harvard). Around 2015, Norbert Kehrer wrapped this emulator into a web-based emulator, which makes the emulator very easy to use (by having a full virtual keyboard to type with and replicates the overall look-and-feel of the IBM 5110 system).

5110VEMU is a 2022 updated desktop build of Christian's original emulator code, to function under the Wintel (Windows/Intel x86 or x64) platform. X11 is replaced by pdcurses, where multiple information windows are presented to provide runtime-state context of the emulator. The intent is to help study runtime behavior between the Executive and Non-Executive portions of the system, in a more observable and controllable fashion.

Setup

This emulator uses the ROS binaries of the IBM 5110. This includes:

execros.bin	Main Executive ROS (handles keyboard input translation and display output updates)
basicros.bin	BASIC ROS (manages execution of the Non-Executive BASIC)
aplros.bin	APL ROS (manages execution of the Non-Executive APL)
comros.bin	Common ROS
basicnx.bin	BASIC Non-Executive (System/3 code)
aplnx.bin	APL Non-Executive (System/370 code)

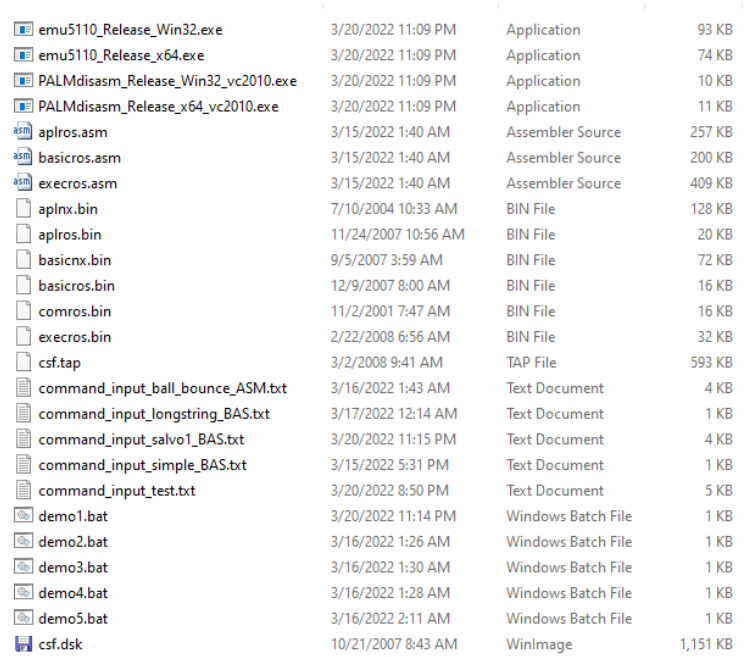
These files are required (using the name specified above) or the emulator will not continue to load.

Optional files are:

csf.dsk	Disk version of the Customer Support Functions (auto-attached on startup)
csf.tap	Tape version of the Customer Support Functions (auto-attached on startup)
(command input)	A text file to script various inputs into the emulator (auto-parsed on startup)

Run the emulator from the bin\ folder, which should also have all the files listed above in that same folder. The current working folder should also be the bin\ (typically this is the case when running programs locally, but this might not be the case if running the emulator from a UNC path like \\fileserver\foo -- use a mapped driver letter or copy the emulator files locally to work around that).

Your overall runtime folder should look like the following:



emu5110_Release_Win32.exe	3/20/2022 11:09 PM	Application	93 KB
emu5110_Release_x64.exe	3/20/2022 11:09 PM	Application	74 KB
PALMdisasm_Release_Win32_vc2010.exe	3/20/2022 11:09 PM	Application	10 KB
PALMdisasm_Release_x64_vc2010.exe	3/20/2022 11:09 PM	Application	11 KB
aplros.asm	3/15/2022 1:40 AM	Assembler Source	257 KB
basicros.asm	3/15/2022 1:40 AM	Assembler Source	200 KB
execros.asm	3/15/2022 1:40 AM	Assembler Source	409 KB
aplnx.bin	7/10/2004 10:33 AM	BIN File	128 KB
aplros.bin	11/24/2007 10:56 AM	BIN File	20 KB
basicnx.bin	9/5/2007 3:59 AM	BIN File	72 KB
basicros.bin	12/9/2007 8:00 AM	BIN File	16 KB
comros.bin	11/2/2001 7:47 AM	BIN File	16 KB
execros.bin	2/22/2008 6:56 AM	BIN File	32 KB
csf.tap	3/2/2008 9:41 AM	TAP File	593 KB
command_input_ball_bounce_ASM.txt	3/16/2022 1:43 AM	Text Document	4 KB
command_input_longstring_BAS.txt	3/17/2022 12:14 AM	Text Document	1 KB
command_input_salvo1_BAS.txt	3/20/2022 11:15 PM	Text Document	4 KB
command_input_simple_BAS.txt	3/15/2022 5:31 PM	Text Document	1 KB
command_input_test.txt	3/20/2022 8:50 PM	Text Document	5 KB
demo1.bat	3/20/2022 11:14 PM	Windows Batch File	1 KB
demo2.bat	3/16/2022 1:26 AM	Windows Batch File	1 KB
demo3.bat	3/16/2022 1:30 AM	Windows Batch File	1 KB
demo4.bat	3/16/2022 1:28 AM	Windows Batch File	1 KB
demo5.bat	3/16/2022 2:11 AM	Windows Batch File	1 KB
csf.dsk	10/21/2007 8:43 AM	WinImage	1,151 KB

HINT: After running the DEMOx.BAT or emu5110_Release_XXX.exe files, a text-console window will appear. If not running full-screen, then use the System Menu (ALT-SPACE) to adjust the Properties to a larger font-size (I use Size 18) to make the console window context easier to read.

Command Line Arguments

You can double-click the emu5110_Release_x64.exe file and the emulator will launch with default settings: BASIC language, no instruction stop, no scripted inputs.

From the command line, you can specify a few behaviors:

EMU5110.EXE <a|b> <stop_at> <input_script>

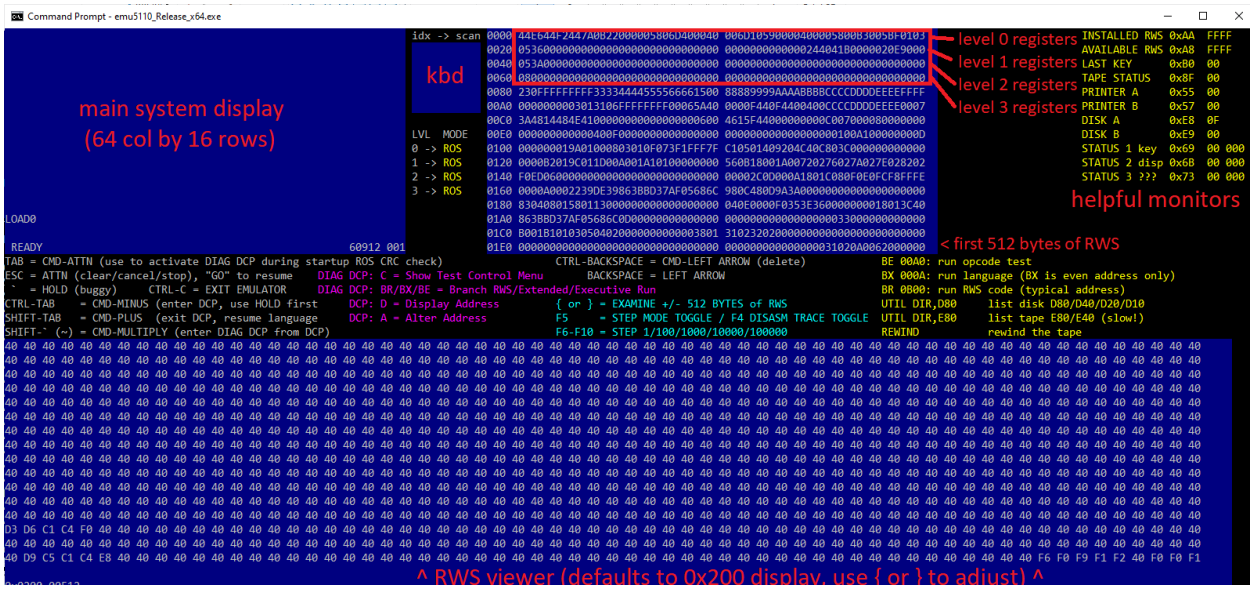
<a|b>	A	APL
	B	BASIC

<stop_at> Enter STEP-MODE when reach instruction count N (use 0 for no-stop)

<input_script> ASCII text filename to be used as scripted inputs

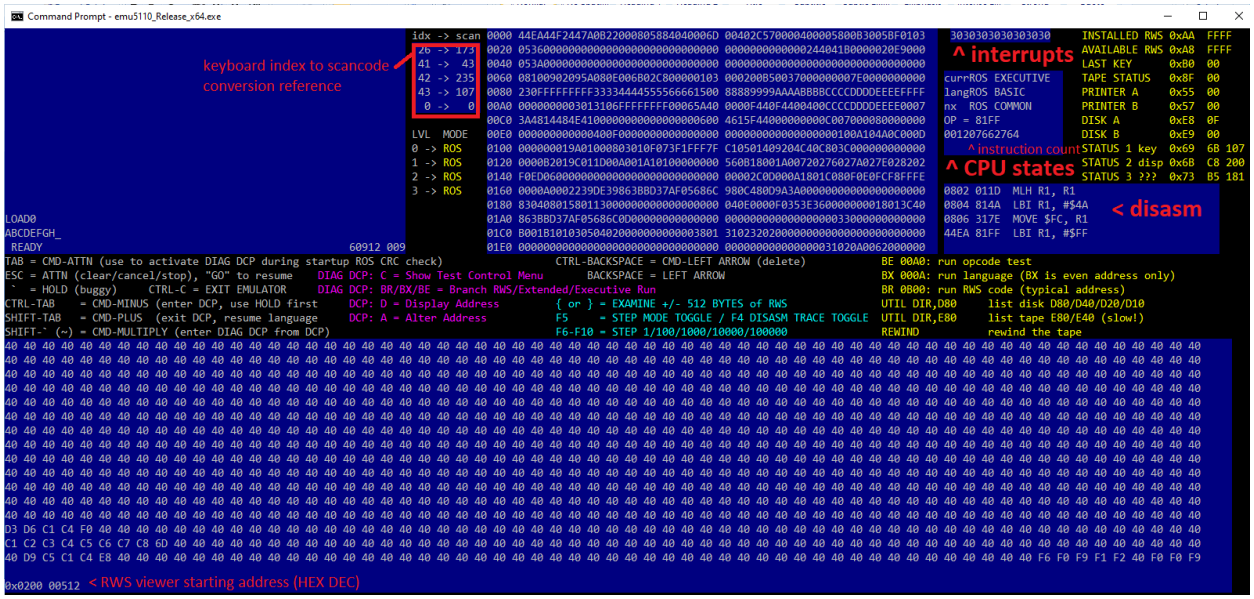
Screen Layout

Here is an overview of the layout of the multiple screen-portions used by 5110VEMU:



When you press F5 to enter STEP MODE, several additional windows appear:

- CPU State Window (MODE state of each interrupt level)
- Disassembler Window (disassembly of the last executed OPCODE)



The “disasm” and “CPU states” update only during TRACE and STEP modes.

Things to Know About the IBM 5110

Main PC Key Mappings

PC KEY (IBM USA)	IBM 5110 Corresponding Scan Code
ENTER/RETURN	EXECUTE
ESCAPE	ATTN (use this to cancel an error, pause runtime, etc.)
` (LEFT-APOSTROPHE)	HOLD (use this first in order to enter DCP mode)
TAB	CMD-ATTN (during ROS initialization at startup, this enters DCP mode)
!	‘ then LEFT-ARROW then . (some characters entered by a composed sequence)
CTRL-TAB	CMD-KEYPAD-MINUS (use this to enter the DCP when on HOLD)
SHIFT-TAB	CMD-KEYPAD-PLUS (use this to exit the DCP back to your language)
SHIFT-` (or ~ TILDE)	CMD-KEYPAD-MULTIPLY (use this to enter DIAG DCP)
CTRL-BACKSPACE	CMD-LEFT ARROW (like pressing DEL to delete current character)
BACKSPACE	LEFT ARROW
?	SHIFT-Q
“	SHIFT-1
&	SHIFT-$
<	SHIFT-3
>	SHIFT-7
:	SHIFT-.
\	SHIFT-/
@	SHIFT-#
(	SHIFT-[
)	SHIFT-]
ARROW KEYS	Mapped to left/right up/down arrow keys
A-Z	A-Z
0-9	0-9
SPACE	SPACE

Emulator Control Keys	
F4	Toggle Disassembler Trace
F5	Toggle Step Mode
F6	Step one instruction
F7	Step 100 instructions
F8	Step 1,000 instructions
F9	Step 10,000 instructions
F10	Step 100,000 instructions
{ or }	Step down/up 512 bytes in RWS monitor window
CTRL-C	Exit the emulator

Presently, there is not much support for mapping PC keyboard inputs into APL symbols and functions. However, APL inputs can be specified using the scripted inputs feature.

Clearing Errors (use ATTN key = press ESCAPE)

The IBM 5110 does not have an ESCAPE key. But it did have an early version of this concept, which it called the ATTN (attention) key. It was located near the number-pad.

After entering a BASIC or APL line of code, it is sent to the "non-executive" to essentially verify that line of code (by parsing it and ensuring it is compatible with the corresponding emulation). Your line of code may be slightly altered during this process (e.g. to fill in defaults). But if the non-executive encounters a parsing error of your line of code, an error code is returned (with a little arrow below the column where the start of the error occurred).

Supported Keyboards

The emulator is doing an IBM PC to IBM 5110 keyboard conversion. Currently the emulator was coded with an English US standard laptop keyboard (i.e. no keypad). You may need to experiment with which keys translate to what on your system/keyboard (i.e. especially for non-US or international keyboards). Unfortunately adapting to a new keyboard style requires a re-compile, no external mapping file has yet been defined.

Registers

The first 128 bytes of RWS is reserved as the system registers for the various execution-levels. These execution-levels are as follows:

LEVEL 0	Normal system execution
LEVEL 1	BSCA and asynchronous communication (not supported in this emulator, no async ROS)
LEVEL 2	Tape, Disk, Printer, Serial I/O (during tape/disk operations, you'll see this level used)
LEVEL 3	Keyboard (you'll see this level used while typing)

Each "level" has 32 registers, which includes the Program Counter as the first register in each level.

Modes

Each "level' also has an operating mode. Conceptually, if you write an interrupt handler, inside that interrupt you could call ROS code (and different types of ROS). You can also run code that is in RWS.

Level 0 has a special startup mode called BUP (basically boot-up processing, or "bring up" mode). This mode is about the first 160,000 instruction cycles of startup. After this startup, then the ROS CRC and startup test are issued (the ABCDEFGHIJKLMNPQ things you see during the startup), which take about another 6,000,000 instructions to complete (where it then examines which programming mode is selected, and proceeds to that corresponding Language ROS).

The system can also be commanded into "RWS mode" where it is executing instructions directly from RWS memory instead of any ROS.

DCP vs DIAG DCP Modes

DCP is Diagnostic Control Program. When in this mode, you will see "DCP" at the top of the screen. This mode has two main features: "A" (ALTER RWS memory) and "D" (display RWS memory). This program is contained in the ROS/ROM of the system.

DIAG DCP is an extended version of DCP that has two additional features: "B" (BRANCH) and "C" (TEST mode). When using any of these commands (A, B, C, D), use ESCAPE (ATTN) to cancel.

Scripted Inputs

Scripted inputs let you emulate pressing a repeatable sequence of keys at startup of the emulator. This is a convenient way to exercise aspects of the system, and return back to specific state-conditions for study.

These inputs are expressed in ASCII text files. The format is as follows:

<command> <parameters> <comments | sequence>

<command> can be

"Q"	Sequence <parameter is ignored, use 0>	[asm]
"A"	ASCII sequence <parameter is ignored, use 0>	[basic]
"C"	CMD-<parameter is index>	
"S"	SHIFT-<parameter is index>	
"N"	Normal key press, <parameter is index>	
"D"	Delay <parameter is number of screen refreshes to wait>	

The difference between "Q" and "A" is that for "A", the newline at the end will be interpreted as to press ENTER (EXECUTE). Therefore, "Q" is generally used for machine-language/assembly inputs, and "A" is generally used for text/BASIC program inputs.

"S", "C", and "N" are single key presses. The specified index is an index into the IBM PC keyboard codes. This means it can be some trial and error to determine which index corresponds to which intended keystroke on the IBM 5110 system. An Excel worksheet is included in the DOCS folder to help maintain this mapping (at least for the laptop keyboard that I am using). While tedious, this input mechanism does technically support non-US keyboards and is also a way to force APL key inputs. This is also a way to do function keys, like CMD-ATTN to enter DCP mode.

Certain inputs may need a pause before the next input, and the "D" Delay command is used to do that. One of the most typical uses is to stall performing inputs during the BUP ("bring up") sequence near the startup of the system. Each screen refresh is approximately 50 instructions, and so a command "D 2" would wait about 100 instructions before proceeding with the next input entry.

See the .TXT files included in the BIN\ folder as examples of input scripts. These scripts are used by the third argument on the command line.

Note that scripted inputs must be parsed to completion before using any other aspect of the emulator. PC keyboard inputs are not enabled during the script input processing. This also means you cannot enter STEP-MODE during the scripted input. Specifying a STOP-AT count on the command line while using a scripted input can lead to some confusion or contradiction, so make sure the STOP-AT count is well beyond however many instructions the scripted input requires.

STEP-MODE and TRACE-MODE

STOP-MODE is where the CPU automatically halts before executing an instruction. The instruction (OPCODE or Operational Code) will be read, but it won't yet be executed. In the physical hardware, there are "cycles" (I and E) to how an opcode is parsed and performed – the emulator does not go exactly to that level of detail.

At the very startup, the STEP-MODE will appear to linger at 0x00A0 twice – which is not exactly accurate. The PC (program counter) has been set during BUP (bring up), but the OPCODE hasn't actually been processed yet. After this is initial STEP, from thereafter the PC address will appear at the "next address" while the OPCODE shown is what is about to be executed (from the previous address).

STEP-MODE is activated in two ways: specifying a non-zero stop-on index in the command line arguments, or pressing F5 during runtime operation of the emulator.

Once in STEP-MODE (toggled by pressing F5), time is given to update the CPU state and disassemble the current opcode. Doing so during normal emulation runtime would slow things down too much, even on modern 3GHz processors (however, this is what the TRACE-MODE does, it enabled updating those two displays during normal runtime of the emulator).

When in STEP-MODE, then use F6 to step one instruction, F7 for 100 instructions, F8 for 1000 instructions, F9 for 10,000 instructions, and F10 for 100,000 instructions – depending on what you are trying to observe. For example, to see individual register changes, you may use F6 to step one instruction at a time. But to "fast forward" to a portion you are more interested in, you might use F7/F8/F9/F10 to do so (in small or larger steps).

TRACE-MODE is toggled by using F4. When activated, this mode forces the update of the CPU state and disassembly window after each instruction. This slows down the emulator significantly, so type slowly as the emulator is technically still interactive while in this mode.

Demos

In the bin\ folder of the emulator are a set of DEMOx.BAT files, to help quickly kick off the emulator in a variety of configurations. This is intended to help "train" on how to use the emulator, and give some ideas on what you can do with it.

DEMO1.BAT – Introduction orientation

This demo replicates the "famous" ball-bounce demo, which is entered in machine language. The emulator does not have an on-the-fly assembler, so while it may be referred to as an "ASM" input, it is really just hard-coded machine instructions. This main point of this demo is to show how to enter the DCP, use the Alter command, enter a sequence of machine instructions, and then execute those instructions.

This is done from a scripted input text file, which can be modified to experiment with various other PALM instructions (or copy to a new text file). Once the script is finished parsing, you can then use F5 to pause (by entering STEP-MODE), then use F6/F7/F8/F9/F10 to step through the instructions (F6 being a single instruction at a time, F10 being 1,000,000 instructions at a time).

Use CTRL-C to exit the emulator. This demo just gets you familiar with a few aspect about the emulator.

DEMO2.BAT – Getting familiar with BASIC inputs

DEMO2 is similar to DEMO1, but inputs a sequence of BASIC code. This does not use the built in DCP of the ROS/ROM. It simulates hand-typing a set of BASIC code directly into the system. It highlights a few important limitations about the IBM 5110 BASIC:

- No double quotes, use single quote (on 5110 keyboard this was SHIFT-K, just use ['] on the IBM PC keyboard)
- No compound statements using colon (:) – you can do compound outputs in a PRINT statement using comma (,) or semicolon (;) but cannot compound-mix PRINT with other statements
    - e.g. PRINT A,B;C is OK
    - e.g. PRINT A:PRINT B:C=A+B:PRINT C is INVALID
- The LET keyword is not supported
- When you LIST, only 16 lines of your program are listed at a time.
    - e.g. if you do "LIST 1000" then the lines 16 prior to line 1000 are listed (including line 1000 itself).
- You can use UP and DOWN arrow to scroll through lines of the program, then edit the line as desired, and press ENTER to commit the changes
    - To erase everything from the current column to the end of the current row, use ESCAPE (which the ATTN key on the original IBM 5110)
- There is no NEW, CLEAR, ERASE (just reset the emulator to clear the current program)
- Use "RUN" to run the current program from the beginning
- While running, you can press ESCAPE (ATTN) to pause the program.
    - Enter the command "GO" to resume the program.

DEMO3.BAT – Small BASIC sample to focus on STEP-MODE usage

This demo enters in a very small BASIC program, that does a calculation, shows (PRINT) a result, then repeats.

Run this program (type "RUN" then press ENTER), then use this "infinite" program to experiment with using F5 to pause/STEP through the program (F6 steps by one instruction, up to F10 to step by 1,000,000 instructions).

Press F5 to exit STEP-MODE and resume the emulation like normal.

DEMO4.BAT – Demonstrate how to use the STOP ON INSTRUCTION command-line feature

For this DEMO4, observe what is in the batch script. The second command line argument lets you automatically enter STEP-MODE at a specific instruction count. Once you've found a situation you're interested, you can script the input and narrow down approximately what instruction count that situation is at. Then from the command line, you can quickly get back to that state, and then proceed with STEP-MODE to examine state behavior.

DEMO5.BAT – Demonstrates how you can enter STEP-MODE right at the first instruction

Similar to DEMO4, this exercises the ENTER-STEP-MODE-ON-INSTRUCTION-N feature of the command line arguments. However, it stops on the very first instruction, the very beginning of the startup sequence of the system.

One special thing to notice here is that LEVEL 0 is in BUP ("bring up") mode. The system is hard coded to boot from the Executive ROS using address 0x000A. The very first instruction is 1FFF (a CTRL instruction to device $F).