Vic20emu User Manual

Table of Contents

Introduction

The Vic-20 Emulator & Debugger focuses on features that help explore the Vic-20 hardware or at least it helped me understanding the machine better than I did 20 years ago. The internal structure of Java program resembles the hardware structure, which is not best for performance, but has other benefits with regard to program analysis and modularity.
If you just want to fire up an emulator and play a few old games there are certainly better and more complete alternatives, e.g. Vice[1] or MESS[2]. Also for writing small programs Vic20 CBM .prg Studio[3] may be provide all you need. Finally, you need a decent PC to run the emulator at reasonable speed.
So why did I feel the urge to write another emulator, if there are so many better alternatives?
First, vic20emu is written in Java, so it supports a wide range of host platforms. It should also be possible to port it to platforms like Android, first results are promising.
Second, I want a modern without a user-interface that is too close to original debuggers on the Vic-20.
Third, I need a debugger that can be started from the development environment with symbols loaded and ready to run my program.

Getting Started

When you start the emulator for the first time, you have to arrange the windows to your needs. Window positions are saved, when you close the application via *Exit* in the file menu (see Figure 1). Select *Run* from the CPU menu and the emulator starts running.

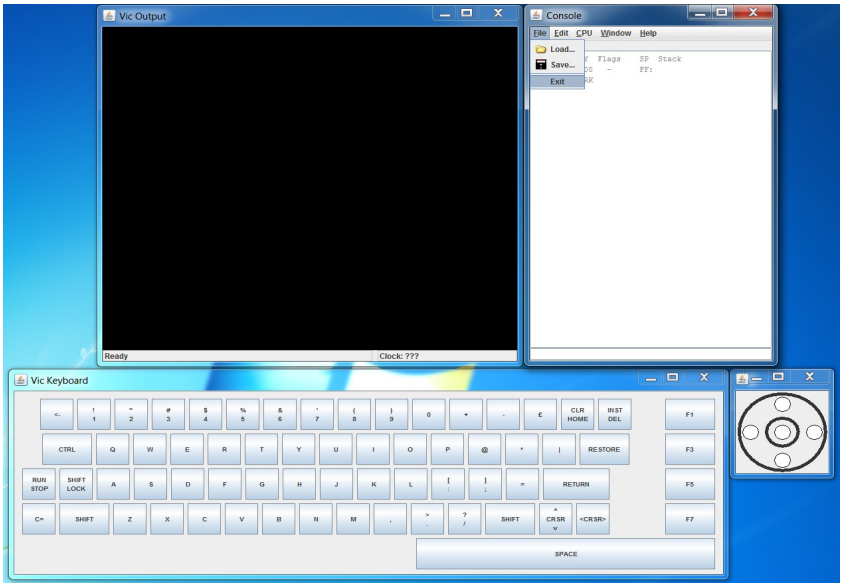


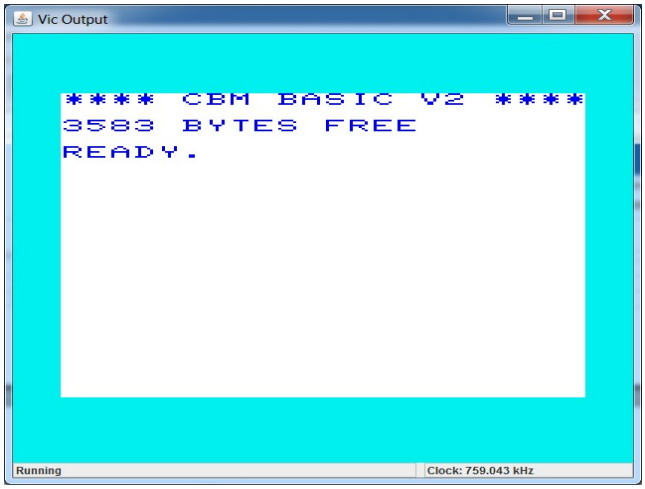
Figure 1: The main windows of the emulator. Close any of the four windows to close the application. Use 'Exit' from the File menu in order to store window positions.

The four main windows of the emulator are the Console window and three peripherals windows for screen, keyboard and joystick.

1 http://www.viceteam.org/
2 http://www.mess.org/
3 http://www.ajordison.co.uk/

Vic Output (screen)

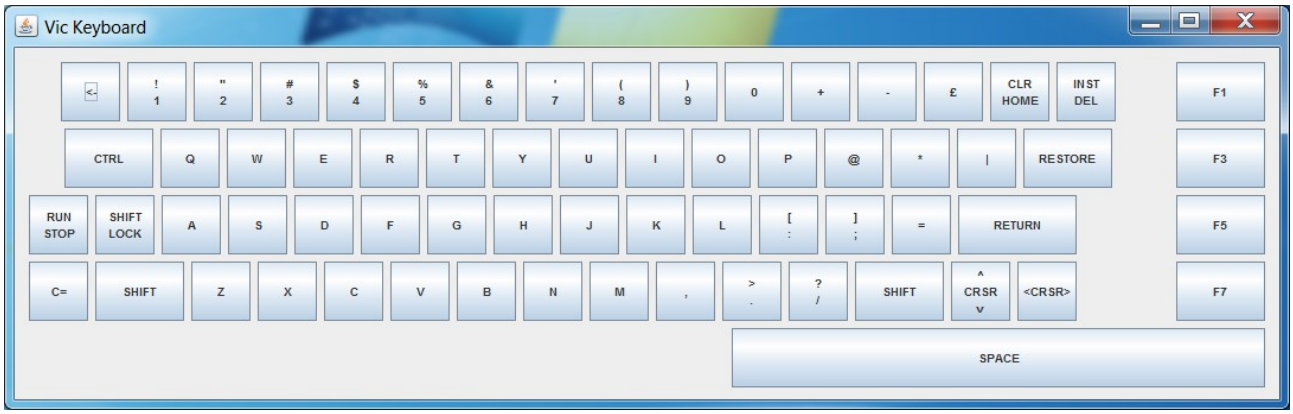
The window displays screen output from the VIC once the CPU is running. The status bar at the bottom of the window shows running state and frequency of the emulated CPU.



You can use your PC keyboard to type text, when the window is active and you can paste text from other sources into the Vic-20 keyboard buffer, by clicking the right mouse button inside the window. The numerical keypad and right Ctrl emulate the Joystick.

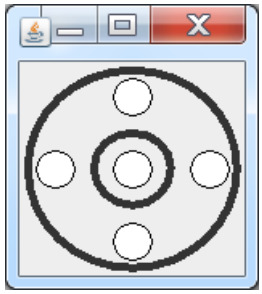
Vic Keyboard

The keyboard window is an imitation of the Vic-20 keyboard. You can use it for keys that are not present on the PC keyboard.



Keys are locked by clicking them with the right mouse button and unlocked by right-clicking them again. This is useful if you want to press multiple keys at the same time or want to freeze keyboard state while hitting a breakpoint.

Vic Joystick

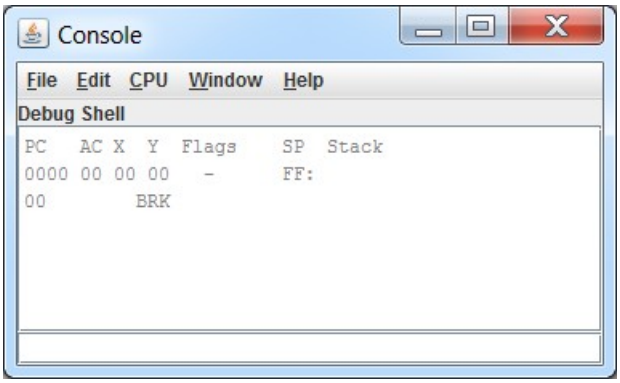


The Joystick window has four small circles for the joystick positions in the outer circle and a circle in the center of the window that shows the state of the fire button.

Move the mouse cursor to the inner circle to "steer" the joystick by the position of the mouse pointer. Click the right mouse button to "fire".

Console

The console window contains menus to access commands and windows and the Debug Shell that shows CPU messages and a command interface (edit field at the bottom). You do not have to use text commands, because for most functions are accessible from the menu or the debugger window.

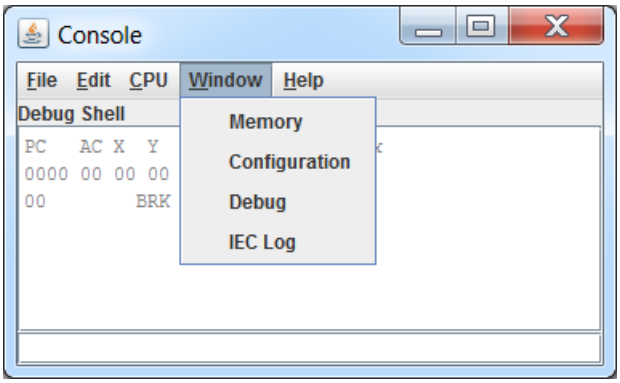


From the **File** menu you can load files into Vic-20 memory or store memory contents to files in different formats.

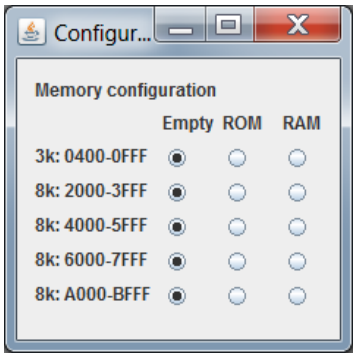
The **Edit** menu is used for command interface input.

The **CPU** menu lets you start, stop and reset the CPU.

The **Window** menu is used to access other useful windows, like configuration, memory, and the debugger window.



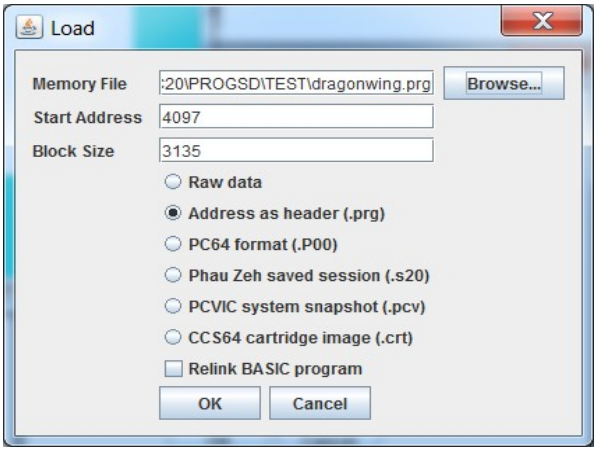
Running a Program



You can load programs directly to memory or use the disk emulation for loading them in the emulator. In both cases you have to select the memory configuration for the program to run. Open the configuration window from the Window menu and set the memory layout appropriately.

Initially all external memory areas are switched off. If you set an area to ROM, the Vic-20 cannot write to it, but you can still load files to memory from the File menu.

The Open command from the file menu lets you select a file from your filesystem and tries to guess start address, size and format, if possible.



For example, the parameters of the file "dragonwing.prg" are guessed correctly after selecting it with the "Browse..." button. The last option "Relink BASIC program" is necessary for BASIC programs that are loaded to a different address.

After loading the program to memory you can switch to the Output window, type RUN and hit Enter to start this great program written by Aleksi Eben. (Unfortunately, the emulator does not play sound yet)

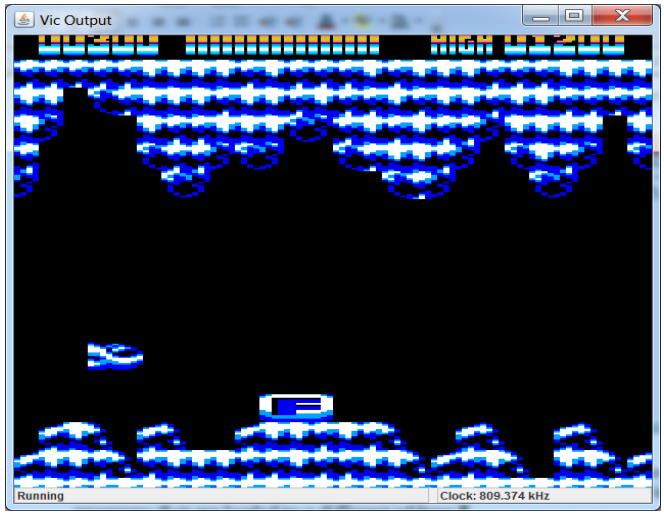
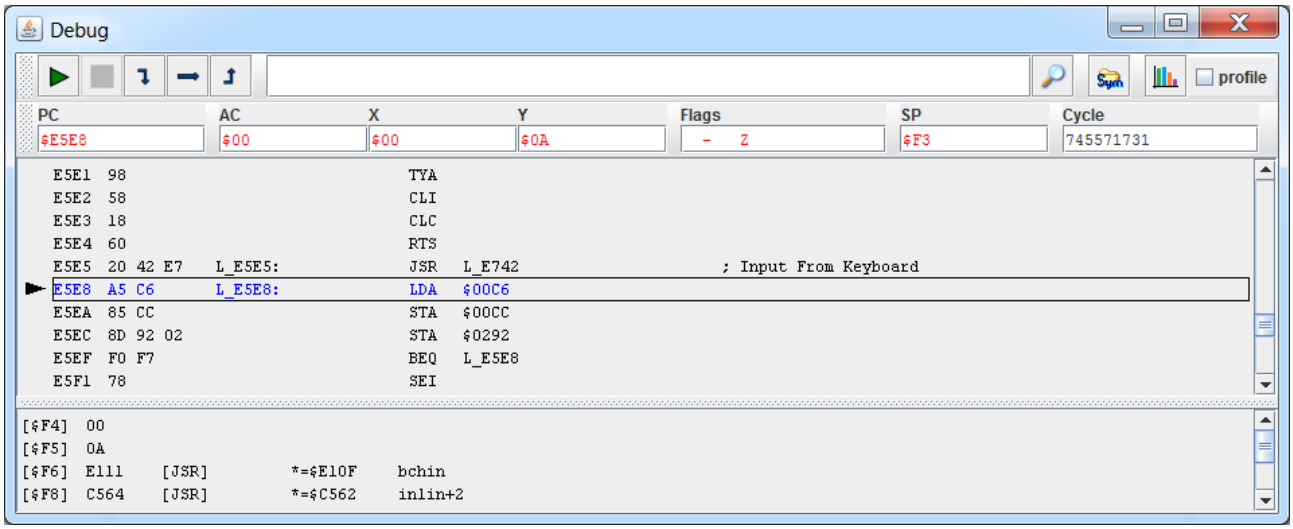


Figure 2: Dragon Wing running in the emulator.

The Debugger Window

You can access the debugger window from the Console (Window menu).

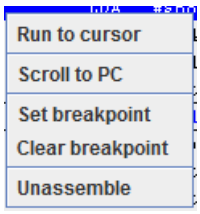


Here you can

- Control execution
- Search for code locations (addresses and labels)
- Load symbols
- Enable profiling and view profiling results
- Review and modify register values
- Browse through code
- Set and clear breakpoints
- Navigate to return addresses on the stack.

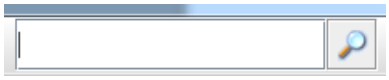
You can start and stop execution as well as single step through the program by using the buttons in the toolbar. The arrow pointing down steps into subroutine calls and interrupts, whereas the straight arrow stops execution after returning from a JSR. The up arrow stops after the next RTS/RTI.

Another way to control execution uses the context menu. Click the right mouse button inside the code area and a menu pops up.



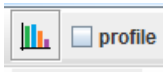
Using this menu you can continue execution until it reaches the selected line, or set and clear breakpoints.

Breakpoints are marked by a red bullet to the left of a code line and can also be set or cleared by a double click in that area. (The black triangle marks the position of the program counter.)

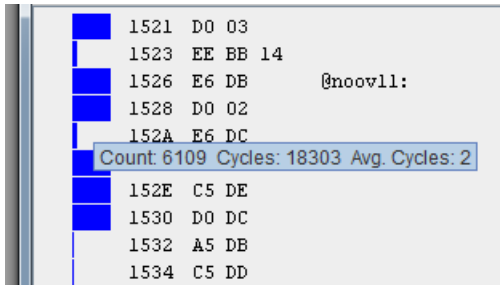


Right next to the command buttons is the search box. You can enter a label or an address and hit enter or the search button to navigate to the corresponding locations in memory. (Hex numbers need a '$' in front, e.g. $FFD2). If the debugger can not find the label or address the search box turns orange.

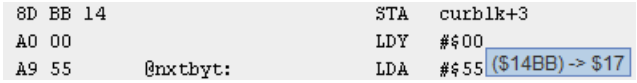
With the "Sym"-Button, you can load symbol files with labels (e.g. from cc65).



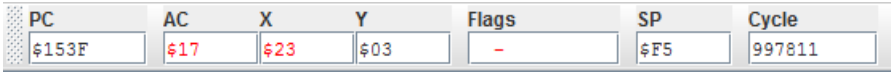
The checkbox labeled "profile" enables execution profiling. When enabled, the debugger displays a blue bar every row in the code window. The width of the bar correlates with how often the statement was executed. Move the mouse pointer over the bar to get more results as a tool tip.



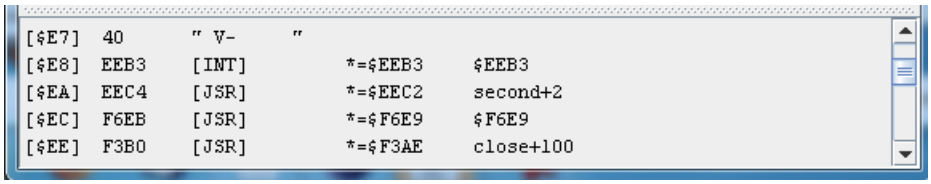
You also get a tool tip when you move the mouse pointer over the operand of a statement. For example, the operand "curblk+3" of the statement STA has the tooltip "($14BB) -> $17.



$14BB is the value of curblk+3 and the target address of the store command STA. The current value at address $14BB is $17. This is useful, if you step through code and want to know what is going on in the next step without looking for the correct locations in the memory window.

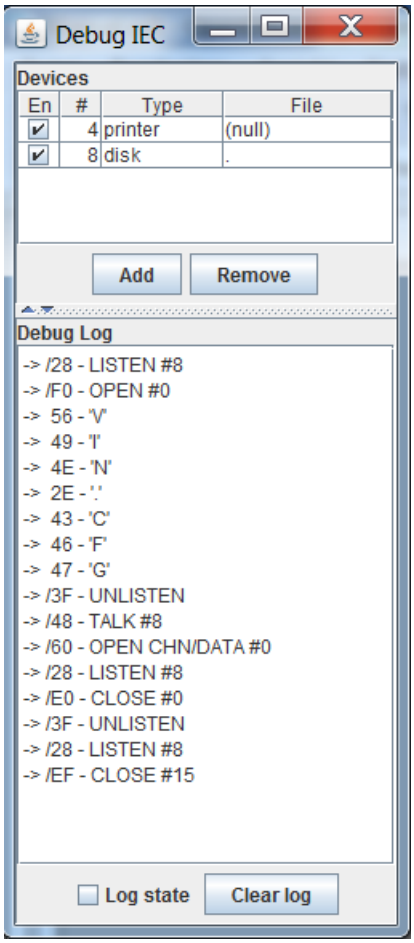


The second toolbar shows register values, stack pointer and CPU cycles elapsed since starting the emulator. Red text color indicates that a value has changed since the last break. You can change each of the values, but not the cycle count.



The stack list in the bottom of the debugger window shows the annotated stack contents. Entries marked with [JSR] or [INT] are pushed by a JSR statement or Interrupt respectively. You can scroll to the corresponding location by double clicking the line in the stack list.

The IEC Window (Debug IEC)



The IEC simulator emulates the communication protocol of Commodore's serial bus. Every byte sent over this bus is logged in the "Debug Log" of the IEC window. Commands bytes are marked by '/' and the commands shown in plain text.

The upper part of the window lists the configured devices on the bus. You can add and remove devices and edit their properties by clicking into the table.

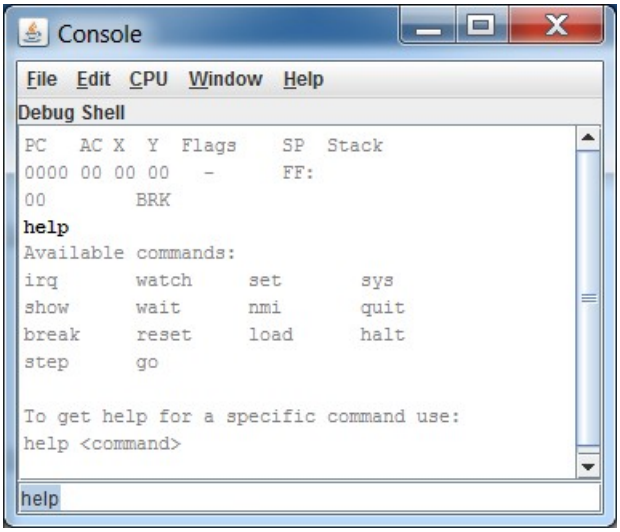
At present, the emulator supports two devices: printer and disk.

Printer is like null device that "consumes" every byte and discards it immediately.

Disk is a file system driver that supports reading files (including "$") from the local file system.

Startup Scripts

You can use the commands of the debugger console to setup the emulator for the program you want to run or debug. To get a list of available commands type 'help' into the command input line at the bottom of the console window.



You get help for a specific command by typing 'help' followed by the command. To run a specific configuration, write the commands into a file and start the emulator with the file name as parameter.

For example, take a look at the startup script I use to debug VIN[4]:

```
go til ready
wait
set ram 1
set ram 2
set ram 3
load code ../vin/diskmenu.prg
load symbols ../vin/diskmenu.lbl
sys prolog
go til prolog
wait
break set panic2
```

'go til' tells the emulator to run until it hits the given address or label. 'ready' is defined in the ROM symbol table (vicrom.sym) as

```
c474    ready   Restart BASIC
```

Since the console continues executing commands after 'go', we have to use the 'wait' command to wait until the CPU stops running. The Vic-20 has finished its initialization and is ready for loading programs. First, the script enables RAM in banks 1-3 ('set ram') and loads the program ('load code') and its symbols from ca65 ('load symbols').
The programs symbol file defines the symbol 'prolog', which is the entry point of the program. The command 'sys' writes the BASIC 'SYS' command to the keyboard buffer and with 'go til' and 'wait' we let the CPU reach 'prolog'.
VIN has an error routine that quits the program and prints register values and stack values. While

4 http://code.google.com/p/vin20/

debugging I prefer to have a breakpoint instead. The last line of the script sets this breakpoint on a location in the error routine ('break set').
After starting the emulator with the script described above, the PC points exactly the entry point of my program and the debugger is ready to run.