The VM Emulator is used for two purposes: (i) Emulating the execution of VM programs on the virtual machine described in the course, and (ii) Executing compiled Jack programs.

A VM program is a set of one or more .vm files, each written in the VM language. VM programs typically result from compiling high-level programs.

The editor panel (labeled ‘VM code’) allows loading a VM program, or writing VM commands directly. The latter is used mainly for experimenting with VM commands and watching their impact on the VM structures and the host RAM.

The VM structures panel visualizes the contents of the main VM data structures: the stack, the call-stack, and the virtual memory segments (local, argument, etc.). The panel displays only the data structures that are relevant to the currently loaded VM code.

The RAM panel displays the host RAM of the Hack computer. There are two panels, both showing the same RAM device and the same address space. Each panel can be scrolled independently.

Loading VM code

When loading a folder from the host file system, the emulator reads all the .vm files in that folder (in no particular order), and creates from them *one* sequence of VM commands. The resulting code base includes all the VM functions in all the .vm files, one after the other (the notion of individual files no longer exists). This code base is then loaded into the emulator. Note: individual files cannot be loaded. Every Jack or VM program must be stored in a folder, even if it includes one file only.

Executing VM code

To run the loaded code interactively from the VM Code panel: Use the step/run/reset controls and the execution speed slider;

To run the loaded code via a test script from the Test panel: Load a test script, and execute it using the step/run/reset controls and the execution speed slider.

Input / Output

If the loaded code includes commands that manipulate the screen, you will see the effects on the simulated screen. If the output / animation is too slow or too fast, reset the code, change the execution speed, and re-run the code.

If the loaded code includes commands that expect to get inputs from the keyboard, click ‘Enable Keyboard’. This will allow entering input values from the physical keyboard of the host device.

RAM panels

This part of the emulator allows exploring and tracking how the virtual machine is realized on the host computer. This is particularly relevant when implementing the VM, as we do in Nand to Tetris projects 7 and 8. For example, you can write VM commands that push some constants onto the stack and perform some arithmetic operations. You can then execute these commands and track the stack pointer (RAM[0]) in the left RAM panel, and the stack itself (RAM[256] onward) in the right RAM panel. The two panels help viewing different parts of the RAM simultaneously.

Bug / issue reports

To report a bug or propose an improvement, click the *bug* icon. You will be asked to login to your GitHub account (if you have one).

www.nand2tetris.org