



# VM Emulator Tutorial

This program is part of the software suite  
that accompanies the book

## ***The Elements of Computing Systems***

by Noam Nisan and Shimon Schocken

MIT Press

[www.nand2tetris.org](http://www.nand2tetris.org)

This software was developed by students at the  
Efi Arazi School of Computer Science at IDC

Chief Software Architects: Yaron Ukrainitz and Yannai Gonczarowski

# Background

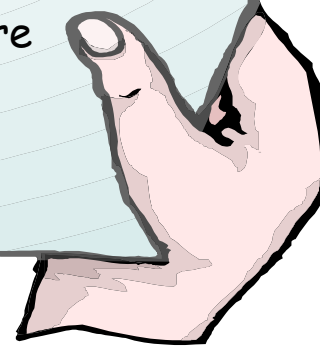
---

*The Elements of Computing Systems* evolves around the construction of a complete computer system, done in the framework of a 1- or 2-semester course.

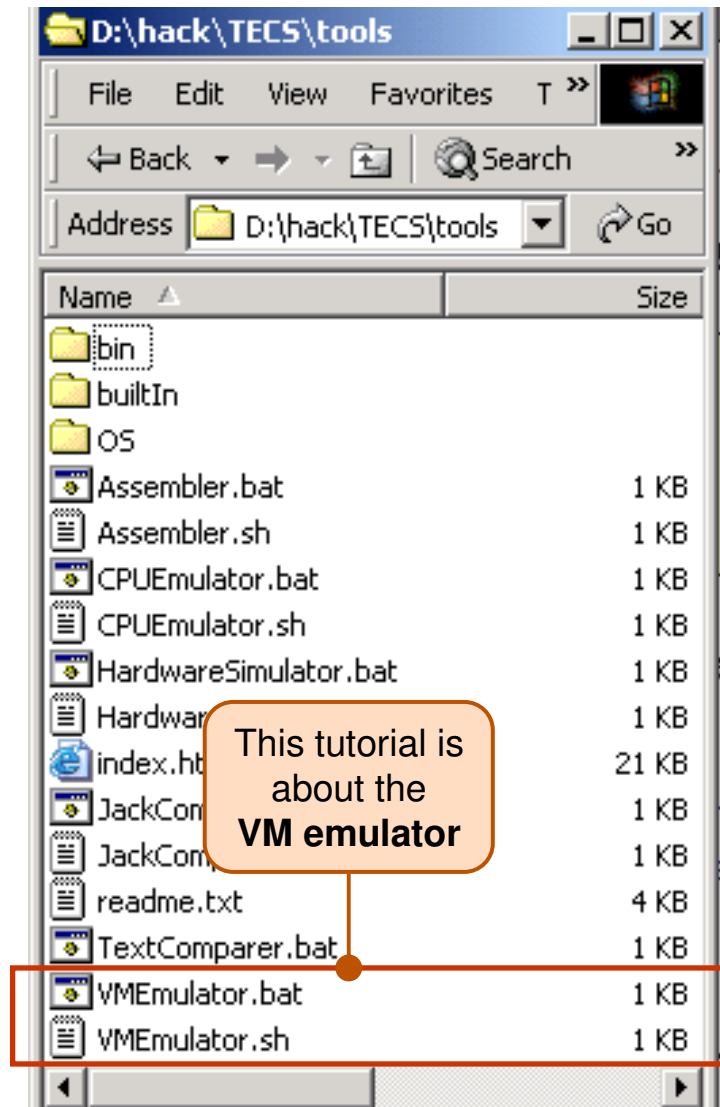
In the first part of the book/course, we build the hardware platform of a simple yet powerful computer, called Hack. In the second part, we build the computer's software hierarchy, consisting of an assembler, a virtual machine, a simple Java-like language called Jack, a compiler for it, and a mini operating system, written in Jack.

The book/course is completely self-contained, requiring only programming as a pre-requisite.

The book's web site includes some 200 test programs, test scripts, and all the software tools necessary for doing all the projects.



# The Book's Software Suite



(All the supplied tools are dual-platform: **xxx.bat** starts **xxx** in Windows, and **xxx.sh** starts it in Unix)

## Simulators

(**HardwareSimulator**, **CPUEmulator**, **VMEulator**):

- Used to build hardware platforms and execute programs;
- Supplied by us.

## Translators (**Assembler**, **JackCompiler**):

- Used to translate from high-level to low-level;
- Developed by the students, using the book's specs; Executable solutions supplied by us.

## Other

- **bin**: simulators and translators software;
- **builtIn**: executable versions of all the logic gates and chips mentioned in the book;
- **os**: executable version of the Jack OS;
- **TextComparer**: a text comparison utility.

# VM Emulator Tutorial

---

- I. [Getting Started](#)
- II. [Using Scripts](#)
- III. [Debugging](#)

Relevant reading (from *The Elements of Computing Systems*):

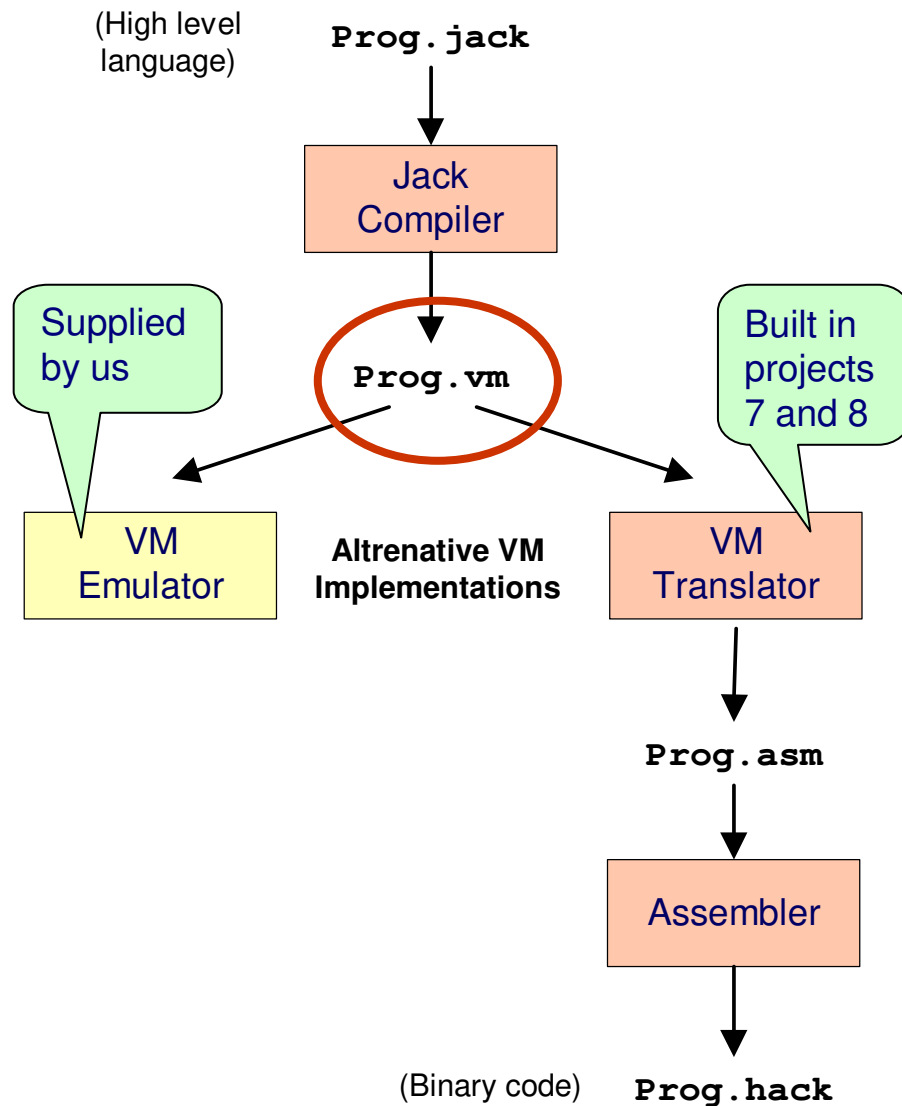
- Chapter 7: *Virtual Machine I: Stack Arithmetic*
- Chapter 8: *Virtual Machine II: Program Control*
- Appendix B: *Test Scripting Language, Section 4.*

# VM Emulator Tutorial

---



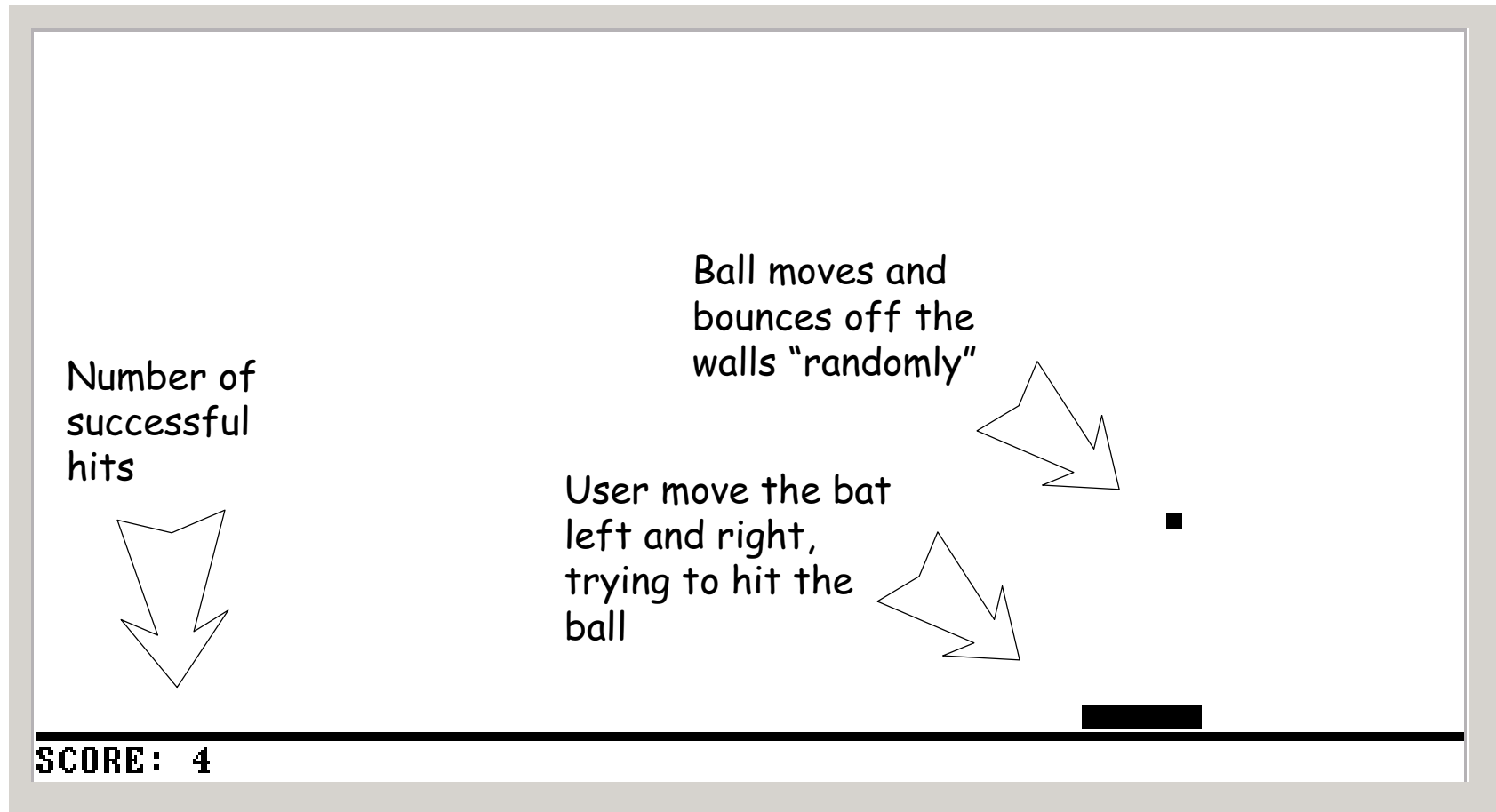
# The Typical Origin of VM Programs



- VM programs are normally written by compilers
- For example, the Jack compiler (chapters 10-11) generates VM programs
- The VM program can be translated further into machine language, and then executed on a host computer
- Alternatively, the same VM program can be emulated as-is on a VM emulator.

## Example: Pong game (user view)

---



Now let's go behind the scene ...

# VM Emulator at a Glance

**Virtual Machine Emulator (1.4b3) - G:\examples\Pong**

**File View Run Help**

**Program**

34	push	local 3
35	push	argument 1
36	lt	
37	not	
38	if-goto	Math.multiply\$WHILE...
39	push	local 3
40	push	static 0
41	add	
42	pop	pointer 1
43	push	that 0
44	push	argument 1
45	and	
46	push	constant 0
47	gt	

**Static**

0	2064
1	2048

**Stack**

3	2064
---	------

**Call Stack**

- Sys.init
- Main.main
- PongGame.run
- Bat.move
- Screen.drawRectangle
- Math.multiply

**Screen:**  
(In this example: Pong game action)

**VM program**  
(In this example: Pong code + OS code)

**The VM emulator serves three purposes:**

- Running programs
- Debugging programs
- Visualizing the VM's anatomy

The emulator's GUI is rather crowded, but each GUI element has an important debugging role.

global stack, as seen by the VM program

**Call stack:**  
Hierarchy of all the functions that are currently running

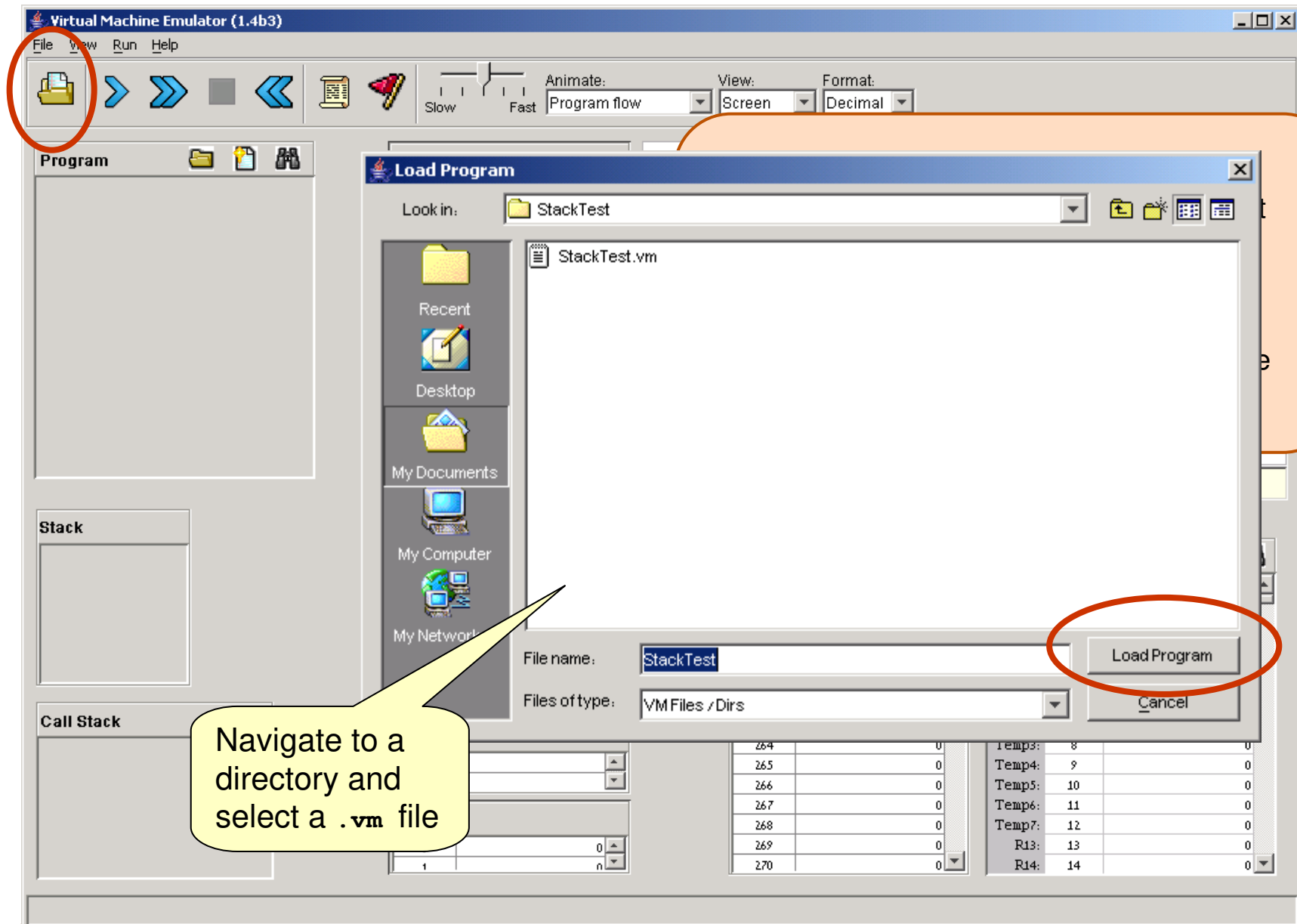
**Not Part of the VM!**  
(displayed in the VM emulator for reference purposes)

**Global stack:**  
Function frames + working stack

**Host RAM:**  
Stores the global stack, heap, etc.



# Loading a VM Program



# Running a Program

The screenshot shows the Virtual Machine Emulator (1.4b3) interface. The title bar indicates the file path: G:\projects\07\StackArithmetic\StackTest\StackTest.vm. The menu bar includes File, View, Run, and Help. The toolbar contains icons for file operations, execution (a red circle highlights the 'Run' button), and animation controls. The 'Program' panel on the left lists 15 instructions. The 'Script' panel on the right shows a default test script. The 'Stack' and 'Call Stack' panels are on the bottom left. The 'Local', 'That', and 'Temp' registers are in the bottom center. The 'RAM' panel on the bottom right shows memory addresses and values. Annotations with yellow callouts provide additional information:

- Script controls:** Points to the execution and animation buttons in the toolbar.
- Default test script:** Points to the script editor, stating: "Always loaded, unless another script is loaded by the user."
- VM code is loaded: (read-only):** Points to the 'Program' panel, stating: "The index on the left is the location of the VM command within the VM code (a GUI effect, not part of the code)."

Index	Op	Value
0	push	constant 17
1	push	constant 17
2	eq	
3	push	constant 892
4	push	constant 891
5	lt	
6	push	constant 32767
7	push	constant 32766
8	gt	
9	push	constant 56
10	push	constant 31
11	push	constant 53
12	add	
13	push	constant 112
14	sub	

```
repeat {
  vmstep;
}
```

SP	Value
0	256

Index	Value
0	0
1	0

Index	Value
0	0
1	0

Index	Value
0	0
1	0

Index	Value
0	0
1	0

Address	Value
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
Temp0	5
Temp1	6
Temp2	7
Temp3	8
Temp4	9
Temp5	10
Temp6	11
Temp7	12
R13	13
R14	14

# Running a Program

Virtual Machine Emulator (1.4b3) - G:\projects\07\StackArithmetic\StackTest\StackTest.vm

File View Run Help

Animate: Program flow View: Script Format: Decimal

Program

Step	Op	Value
0	push	constant 17
1	push	constant 17
2	eq	
3	push	constant 892
4	push	constant 891
5	lt	
6	push	constant 32767
7	push	constant 32766
8	gt	
9	push	constant 56
10	push	constant 31
11	push	constant 53
12	add	
13	push	constant 112
14	sub	

Stack

Index	Value
-1	
0	
-1	
56	
84	

Call Stack

Static

Index	Value
0	
1	
2	
3	
4	

Local

Index	Value
0	0
1	0
2	0
3	0
4	0

Argument

Index	Value
0	0
1	0
2	0
3	0
4	0

This

Index	Value
0	0
1	0
2	0
3	0
4	0

That

Index	Value
0	0
1	0
2	0
3	0
4	0

repeat {  
 vmstep;  
}

Global Stack

Index	Value
256	-1
257	0
258	-1
259	56
260	84
261	53
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

RAM

Index	Value
SP: 0	261
LCL: 1	0
ARG: 2	0
THIS: 3	0
THAT: 4	0
Temp0: 5	0
Temp1: 6	0
Temp2: 7	0
Temp3: 8	0
Temp4: 9	0
Temp5: 10	0
Temp6: 11	0
Temp7: 12	0
R13: 13	0
R14: 14	0

Impact of first 13 "vmsteps"

# Loading a Multi-File Program

Virtual Machine Emulator (1.4b1)

File View Run Help

Program

Static

0	
1	
2	
3	
4	

Local

0	256
1	0
2	0
3	0
4	0

Argument

0	256
1	0

Working Stack

Call Stack

Temp

0	0
1	0

Load Program

Look in: Pong

- Array.vm
- Ball.vm
- Bat.vm
- Keyboard.vm
- Main.vm

File name: Pong

Files of type: VM Files / Dirs

Load Program

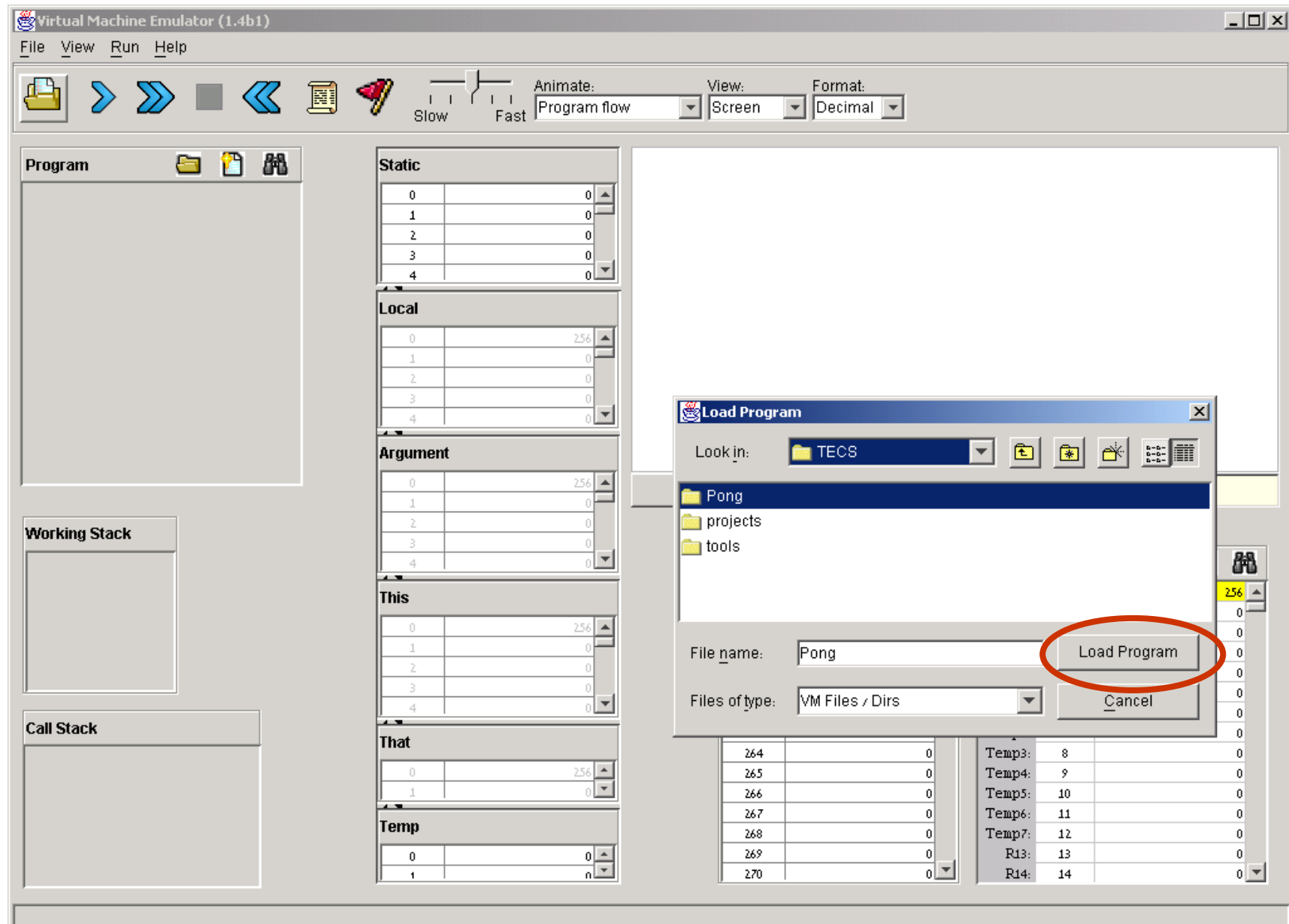
Cancel

**Won't work!**

Why? Because Pong is a multi-file program, and ALL these files must be loaded. Solution: navigate back to the directory level, and load it.

- Most VM programs, like Pong, consist of more than one `.vm` file. For example, the Jack compiler generates one `.vm` file for each `.jack` class file, and then there are all the `.vm` files comprising the operating system. All these files must reside in the same directory.
- Therefore, when loading a multi-file VM program into the VM emulator, one must load the *entire directory*.

# Loading a Multi-File Program





# Virtual Memory Segments

**Virtual Machine Emulator (1.4b3) - G:\examples\Pong**

File View Run Help

Animate: Program flow View: Screen Format: Screen

Slow Fast

**Program**

93	push	local 0
94	push	constant 1
95	add	
96	pop	local 0
	label	Math.divide\$IF_FAL...
	label	Math.divide\$IF_FAL...
97	goto	Math.divide\$WHILE_...
	label	Math.divide\$WHILE_...
	label	Math.divide\$WHILE_...
98	push	local 0
99	push	constant 1
100	neg	
101	gt	
102	not	
103	if-goto	Math.divide\$WHILE_...

**Static**

0	2064
1	2048

**Local**

0	4
1	0
2	0
3	-1

**Argument**

0	361
1	16

**This**

0	362
1	229
2	50
3	7
4	2

**That**

0	512
1	0

**Temp**

0	512
1	n

**Memory segments:**

- The VM emulator displays the states of 6 of the 8 VM's memory segments;
- The **Constant** and **Pointer** segments are not displayed.

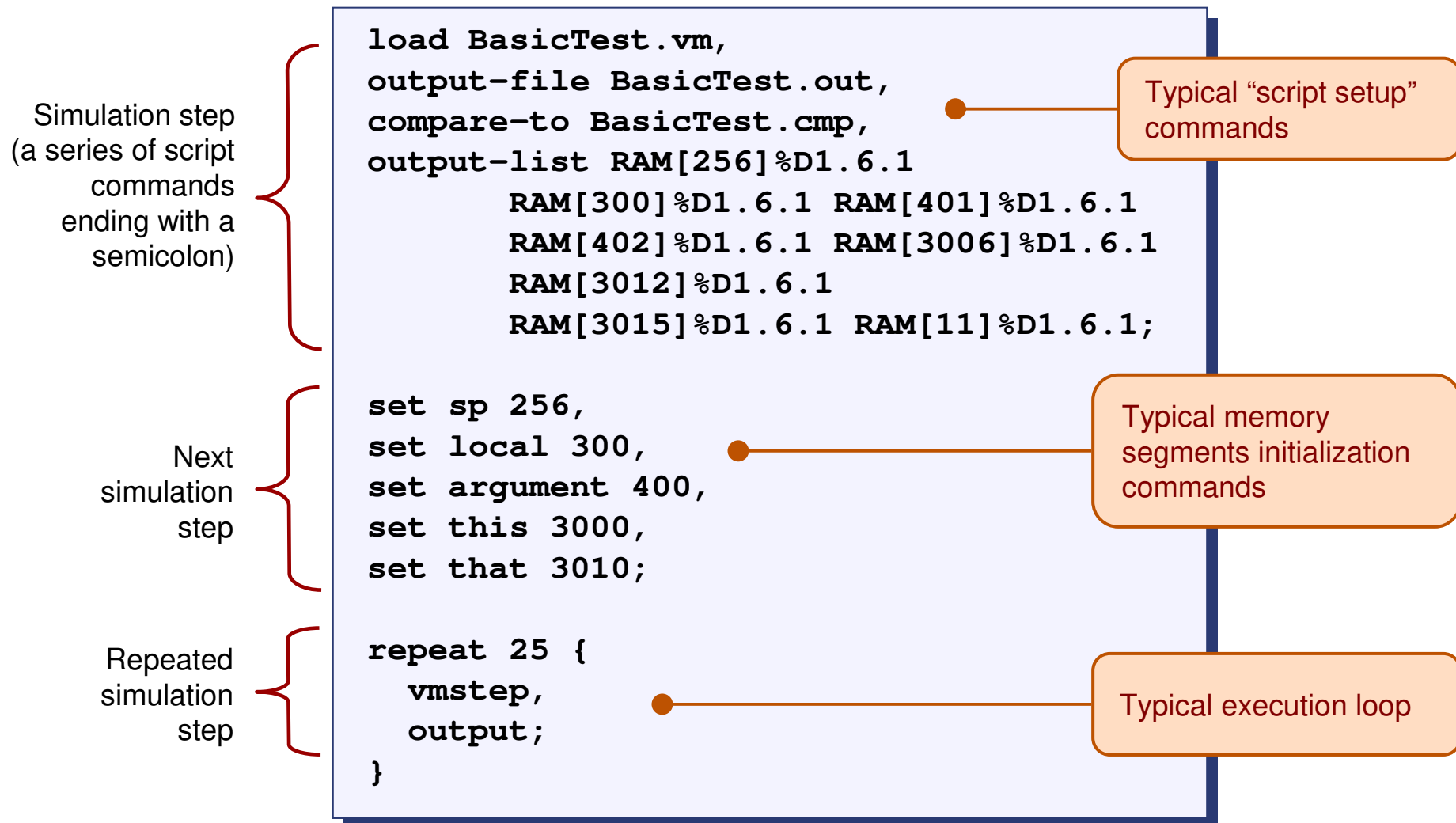
**A technical point to keep in mind:**

- Most VM programs include **pop** and **push** commands that operate on **Static**, **Local**, **Argument**, etc.;
- In order for such programs to operate properly, VM implementations must initialize the memory segments' bases, e.g. anchor them in selected addresses in the host RAM;
- Case 1: the loaded code includes function calling commands. In this case, the VM implementation takes care of the required segment initializations in run-time, since this task is part of the VM function call-and-return protocol;
- Case 2: the loaded code includes no function calling commands. In this case, the common practice is to load the code through a *test script* that handles the necessary initialization externally.

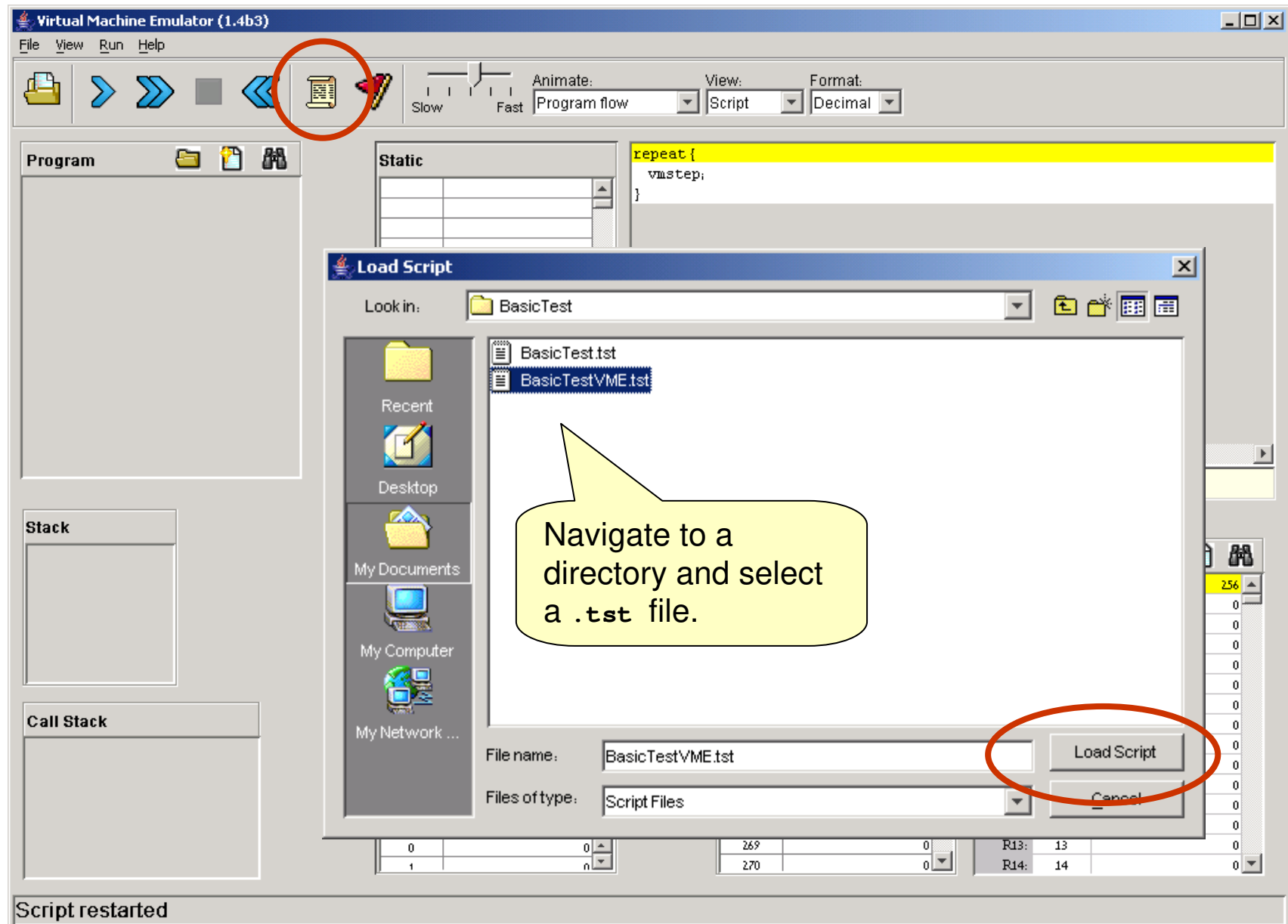




# Typical VM Script



# Loading a Script



# Script Controls

The screenshot shows the Virtual Machine Emulator (1.4b3) interface. The top menu bar includes File, View, Run, and Help. Below the menu is a toolbar with icons for file operations and execution. The main window is divided into several panels: Program, Static, Stack, Call Stack, and a central script editor. The script editor contains the following code:

```
load BasicTest.vm,  
output-file BasicTest.out,  
compare-to BasicTest.cmp,  
output-list RAM[256]%D1.6.1 RAM[300]%D1.6.1 RAM[401]%D1.6.1  
RAM[402]%D1.6.1 RAM[3006]%D1.6.1 RAM[3012]%D1.6.1  
RAM[3015]%D1.6.1 RAM[11]%D1.6.1;  
  
set sp 256,  
set local 300,  
set argument 400,  
set this 3000,  
set that 3010,  
  
repeat 25 {  
vmstep;  
}
```

Callouts point to specific controls:

- Execution speed control:** Points to the 'Animate' dropdown menu, which is set to 'Program flow'. Below it are 'Slow' and 'Fast' buttons.
- Reset the script:** Points to the 'Program' panel icon (a document with a magnifying glass).
- Pause the simulation:** Points to the 'Run' button (a play icon).
- Execute step after step repeatedly:** Points to the 'Single Step' button (a single arrow icon).
- Execute the next simulation step:** Points to the 'Next Step' button (a double arrow icon).

The 'Static' panel shows a table with columns for address, value, and a small icon. The 'Stack' panel shows a table with columns for address, value, and a small icon. The 'Call Stack' panel shows a table with columns for address, value, and a small icon. The 'Global Stack' panel shows a table with columns for address, value, and a small icon. The 'RAM' panel shows a table with columns for address, value, and a small icon.

Script = a series of simulation steps, each ending with a semicolon;

New script loaded: G:\projects\07\MemoryAccess\BasicTest\BasicTestVME.tst

# Running the Script

The screenshot shows the Virtual Machine Emulator (1.4b3) interface. The 'Run' button, represented by a blue right-pointing arrow, is circled in red. A callout box with an orange border and a red dot pointing to the 'Run' button contains the text: "Loads a VM program into the emulator".

The interface includes a menu bar (File, View, Run, Help) and a toolbar with icons for file operations and execution. The main window is divided into several panels:

- Program:** A large empty area for the loaded program.
- Static:** A table for static variables.
- Local:** A table for local variables.
- Argument:** A table for arguments.
- This:** A table for 'this' pointers.
- That:** A table for 'that' pointers.
- Temp:** A table for temporary variables.
- Global Stack:** A table showing the global stack with addresses from 256 to 270.
- RAM:** A table showing RAM memory with addresses from 0 to 14.
- Stack:** A table for the current stack.
- Call Stack:** A table for the call stack.

The script loaded is `load BasicTest.vm`, which includes instructions for output, comparison, list generation, and a loop.

New script loaded: G:\projects\07\MemoryAccess\BasicTest\BasicTestVME.tst

# Running the Script

The screenshot shows the Virtual Machine Emulator (1.4b3) interface. The title bar indicates the file path: G:\projects\07\MemoryAccess\BasicTest\BasicTest.vm. The menu bar includes File, View, Run, and Help. The toolbar contains icons for file operations and execution, with the 'Run' button (a blue arrow) circled in red. Below the toolbar, the 'Program' list shows 14 instructions, with the first instruction 'push constant 10' highlighted. A yellow callout bubble points to the 'Run' button with the text 'VM code is loaded'. The right panel displays the script code, with the line 'set sp 256,' highlighted. The bottom section shows the 'Static', 'Local', 'Argument', 'This', 'That', and 'Temp' memory spaces, along with the 'Global Stack' and 'RAM' memory spaces.

**Program**

Index	Operation	Value
0	push	constant 10
1	pop	local 0
2	push	constant 21
3	push	constant 22
4	pop	argument 2
5	pop	argument 1
6	push	constant 36
7	pop	this 6
8	push	constant 42
9	push	constant 45
10	pop	that 5
11	pop	that 2
12	push	constant 510
13	pop	temp 6
14	push	local 0

**Static**

Index	Value
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

**Local**

Index	Value
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

**Argument**

Index	Value
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

**This**

Index	Value
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

**That**

Index	Value
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

**Temp**

Index	Value
0	0
1	0

**Global Stack**

Index	Value
256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

**RAM**

Index	Value
SP: 0	256
LCL: 1	0
ARG: 2	0
THIS: 3	0
THAT: 4	0
Temp0: 5	0
Temp1: 6	0
Temp2: 7	0
Temp3: 8	0
Temp4: 9	0
Temp5: 10	0
Temp6: 11	0
Temp7: 12	0
R13: 13	0
R14: 14	0

**Script Code**

```
load BasicTest.vm,  
output-file BasicTest.out,  
compare-to BasicTest.cmp,  
output-list RAM[256] %D1.6.1 RAM[300] %D1.6.1 RAM[401] %D1.6.1  
          RAM[402] %D1.6.1 RAM[3006] %D1.6.1 RAM[3012] %D1.6.1  
          RAM[3015] %D1.6.1 RAM[11] %D1.6.1;  
  
set sp 256,  
set local 300,  
set argument 400,  
set this 3000,  
set that 3010,  
  
repeat 25 {  
  vmstep;  
}
```

# Running the Script

(click a few times)

The memory segments were initialized (their base addresses were anchored to the RAM locations specified by the script).

A loop that executes the loaded VM program

**Program**

Index	Op	Value
0	push	constant 10
1	pop	local 0
2	push	constant 21
3	push	constant 22
4	pop	argument 2
5	pop	argument 1
6	push	constant 36
7	pop	this 6
8	push	constant 42
9	push	constant 45
10	pop	that 5
11	pop	that 2
12	push	constant 510
13	pop	temp 6
14	push	local 0

**Static**

Index	Value
0	0
1	0
2	0
3	0
4	0

**Local**

Index	Value
0	0
1	0
2	0
3	0
4	0

**Argument**

Index	Value
0	0
1	0
2	0
3	0
4	0

**This**

Index	Value
0	0
1	0
2	0
3	0
4	0

**That**

Index	Value
0	0
1	0

**Temp**

Index	Value
0	0
1	0

**Global Stack**

Index	Value
256	10
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

**RAM**

Index	Value
SP: 0	257
LCL: 1	300
ARG: 2	400
THIS: 3	3000
THAT: 4	3010
Temp0: 5	0
Temp1: 6	0
Temp2: 7	0
Temp3: 8	0
Temp4: 9	0
Temp5: 10	0
Temp6: 11	0
Temp7: 12	0
R13: 13	0
R14: 14	0

```
output-list RAM[256] %D1.6.1 RAM[300] %D1.6.1 RAM[401] %D1.6.1
RAM[402] %D1.6.1 RAM[3006] %D1.6.1 RAM[3012] %D1.6.1
RAM[3015] %D1.6.1 RAM[11] %D1.6.1;

set sp 256,
set local 300,
set argument 400,
set this 3000,
set that 3010,

repeat 25 {
  vmstep;
}

output;
```

# Running the Script

Virtual Machine Emulator (1.4b3) - G:\projects\07\MemoryAccess\BasicTest\BasicTest.vm

File View Run Help

Slow Fast Animate: Program flow View: Script Format: Decimal

**Program**

0	push	constant 10
1	pop	local 0
2	push	constant 21
3	push	constant 22
4	pop	argument 2
5	pop	argument 1
6	push	constant 36
7	pop	this 6
8	push	constant 42
9	push	constant 45
10	pop	that 5
11	pop	that 2
12	push	constant 510
13	pop	temp 6
14	push	local 0

**Static**



**Local**

0	10
1	0
2	0
3	0
4	0

**Argument**

0	0
1	21
2	22
3	0
4	0

**This**

2	0
3	0
4	0
5	0
6	36

**That**

0	0
1	0

**Temp**

0	0
1	0

**Stack**

	42
	45

**Global Stack**

256	42
257	45
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

**RAM**

SP:	0	258
LCL:	1	300
ARG:	2	400
THIS:	3	3000
THAT:	4	3010
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

output-list RAM[256]%D1.6.1 RAM[300]%D1.6.1 RAM[401]%D1.6.1  
RAM[402]%D1.6.1 RAM[3006]%D1.6.1 RAM[3012]%D1.6.1  
RAM[3015]%D1.6.1 RAM[11]%D1.6.1;

set sp 256,  
set local 300,  
set argument 400,  
set this 3000,  
set that 3010,

repeat 25 {  
vmstep;  
}

output;

Impact after first 10 commands are executed





# View Options

The screenshot shows the Virtual Machine Emulator (1.4b1) interface. The title bar indicates the file path: G:\TECS\projects\07\MemoryAccess\BasicTest\BasicTest.vm. The menu bar includes File, View, Run, and Help. The toolbar contains icons for file operations and execution, along with an 'Animate' slider and 'View' and 'Format' dropdowns. The 'View' dropdown is currently set to 'Script'. The 'Format' dropdown is set to 'Decimal'. The main window is divided into several panes: 'Program' (line 10 to 24), 'Static' (addresses 0 to 4), 'Local' (addresses 0 to 4), 'Argument' (addresses 0 to 4), 'This' (address 0), 'Temp' (addresses 5 to 6), 'Working Stack' (address 472), and 'Call Stack'. A large yellow callout bubble points to the 'View' dropdown, containing the text 'View options:' followed by a list of options: 'Script', 'Output', 'Compare', and 'Screen'. The 'Script' option is selected. The 'Output' pane shows the generated output file, which includes a comparison report. The 'Compare' pane shows the given comparison file. The 'Screen' pane shows the simulated screen. The status bar at the bottom indicates 'End of script - Comparison ended successfully'.

When the script terminates, the comparison of the script output and the compare file is reported.

**View options:**

- **Script**: displays the loaded script;
- **Output**: displays the generated output file;
- **Compare**: displays the given comparison file;
- **Screen**: displays the simulated screen.

End of script - Comparison ended successfully

# Animation Options

The screenshot shows the Virtual Machine Emulator (1.4b1) interface. The title bar reads "Virtual Machine Emulator (1.4b1) - G:\TECS\Pong". The menu bar includes "File", "View", "Run", and "Help". The toolbar contains icons for file operations and execution controls, including a speed slider between "Slow" and "Fast". The "Animate:" dropdown is set to "Program & data flow", "View:" is set to "Screen", and "Format:" is set to "Decimal".

The main window is divided into several panels:

- Program:** A list of assembly instructions. Line 64, "add", is highlighted in yellow.
- Static:** A table of static variables.
- Argument:** A table of arguments.
- This:** A table of 'this' pointers.
- Working Stack:** A stack of values, with 3664 at the top.
- Call Stack:** A list of active function calls, including "Sys.init", "Main.main", "PongGame.run", "Bat.move", "Screen.drawRectangle", and "Math.multiply".

Annotations with callouts explain various features:

- Speed control (of both execution and animation):** Points to the speed slider.
- source:** Points to the "Argument" table.
- transit:** Points to the "Working Stack" table.
- destn.:** Points to the "Working Stack" table.
- data flow animation related to the last VM command (in this example: push argument 0):** Points to the "Working Stack" table.

**Animation control:**

- **Program flow** (default): highlights the next VM command to be executed;
- **Program & data flow:** highlights the next VM command and animates data flow;
- **No animation:** disables all animation

**Usage tip:** To execute any non-trivial program quickly, select *no animation*.

# Breakpoints: a Powerful Debugging Tool

---

The VM emulator keeps track of the following variables:

- **segment[i]**: Where segment is either **local**, **argument**, **this**, **that**, or **temp**
- **local**, **argument**, **this**, **that**: Base addresses of these segments in the host RAM
- **RAM[i]**: Value of this memory location in the host RAM
- **sp**: Stack pointer
- **currentFunction**: Full name (inc. fileName) of the currently executing VM function
- **line**: Line number of the currently executing VM command

Breakpoints:

- A breakpoint is a pair *<variable, value>* where *variable* is one of the labels listed above (e.g. **local[5]**, **argument**, **line**, etc.) and *value* is a valid value
- Breakpoints can be declared either interactively, or via script commands
- For each declared breakpoint, when the *variable* reaches the *value*, the emulator pauses the program's execution with a proper message.

The screenshot shows the Virtual Machine Emulator (1.4b3) interface. The main window displays a program listing on the left, a static panel in the center, and a breakpoint variables panel on the right. The program listing shows a simple VM program with instructions like push, pop, add, sub, return, call, label, and goto. The static panel shows the current function header and the next command. The breakpoint variables panel shows the current function and the variable to be broken on. The interface includes a menu bar (File, View, Run, Help), a toolbar with icons for file operations, execution, and breakpoints, and a status bar at the bottom.

Annotations and steps:

1. Open the breakpoint panel
2. Previously-declared breakpoints
3. Add, delete, or update breakpoints
4. Select the variable on whose value you wish to break
5. Enter the value at which the break should occur

Additional notes:

- By convention, function headers are colored violet
- Here the violet coloring is overridden by the yellow "next command" highlight.

A simple VM program: **Sys.init** calls **Main.main**, that calls **Main.add** (header not seen because of the scroll), that does some simple stack arithmetic.

# Setting Breakpoints

**Breakpoints logic:**  
When `local[1]` will become 8, or when `sp` will reach 271, or when the command in line 13 will be reached, or when execution will reach the `Main.add` function, the emulator will pause the program's execution.

**Program**

Line	Command	Target
9	push	local 0
10	push	local 1
11	add	
12	pop	local 0
13	push	local 1
14	push	local 0
15	sub	
16	return	
0	function	Main.main 0
1	call	Main.add 0
2	return	
0	function	Sys.init 0
1	call	Main.main 0
	label	Sys.init\$INFINITELOOP
2	goto	Sys.init\$INFINITELOOP

**Breakpoint Panel**

Variable Name	Value
local[1]	8
sp	271
line	13
currentFunction	Main.add

**Stack**

**Call Stack**

**This**

**That**

**Temp**

Temp	Value
0	0
1	0

**RAM**

Address	Value
SP: 0	256
LCL: 1	0
ARG: 2	0
THIS: 3	0
THAT: 4	0
Temp0: 5	0
Temp1: 6	0
Temp2: 7	0
Temp3: 8	0
Temp4: 9	0
Temp5: 10	0
Temp6: 11	0
Temp7: 12	0
R13: 13	0
R14: 14	0

# Breakpoints in Action

Virtual Machine Emulator (1.4b3) - G:\examples\add

File View Run Help

Animate: Program flow View: Script Format: Decimal

Program

Address	Function	Code
0	function	Main.add 3
1	push	constant 15
2	pop	local 0
3	push	constant 7
4	pop	local 1
5	push	local 1
6	push	constant 1
7	add	
8	pop	local 1
9	push	local 0
10	push	local 1
11	add	
12	pop	local 0
13	push	local 1
14	push	local 0

Static

Address	Value
0	
1	
2	

Local

Address	Value
0	0
1	0
2	0

Argument

Address	Value
0	
1	
2	

Stack

Call Stack

Function
Sys.init
Main.main
Main.add

Breakpoint Panel

Variable Name	Value
local[1]	8
sp	271
line	13
currentFunction	Main.add

RAM

Address	Value
SP: 0	269
LCL: 1	266
ARG: 2	261
THIS: 3	0
THAT: 4	0
Temp0: 5	0
Temp1: 6	0
Temp2: 7	0
Temp3: 8	0
Temp4: 9	0
Temp5: 10	0
Temp6: 11	0
Temp7: 12	0
R13: 13	0
R14: 14	0

Temp

Address	Value
0	0
1	0

Execution reached the **Main.add** function, an event that triggers a display of the breakpoint and execution pause.

Breakpoint reached

# Breakpoints in Action

Virtual Machine Emulator (1.4b3) - G:\examples\add

File View Run Help

Animate: Program flow View: Script Format: Decimal

Program

Index	Function	Instruction
0	function	Main.add 3
1	push	constant 15
2	pop	local 0
3	push	constant 7
4	pop	local 1
5	push	local 1
6	push	constant 1
7	add	
8	pop	local 1
9	push	local 0
10	push	local 1
11	add	
12	pop	local 0
13	push	local 1
14	push	local 0

Static

Index	Value
0	
1	
2	

Local

Index	Value
0	15
1	7
2	0

Argument

Index	Value
0	
1	
2	

Stack

Index	Value
7	
1	

Call Stack

Function
Sys.init
Main.main
Main.add

Breakpoint Panel

Variable Name	Value
local[1]	8
sp	271
line	13
currentFunction	Main.add

RAM

Index	Value
SP: 0	271
LCL: 1	266
ARG: 2	261
THIS: 3	0
THAT: 4	0
Temp0: 5	0
Temp1: 6	0
Temp2: 7	0
Temp3: 8	0
Temp4: 9	0
Temp5: 10	0
Temp6: 11	0
Temp7: 12	0
R13: 13	0
R14: 14	0

Breakpoint reached

Following some **push** and **pop** commands, the stack pointer (**sp**) became 271, an event that triggers a display of the breakpoint and execution pause.

# Breakpoints in Action

Virtual Machine Emulator (1.4b3) - G:\examples\add

File View Run Help

Animate: Program flow View: Script Format: Decimal

Program

Line	Function	Instruction
0	function	Main.add 3
1	push	constant 15
2	pop	local 0
3	push	constant 7
4	pop	local 1
5	push	local 1
6	push	constant 1
7	add	
8	pop	local 1
9	push	local 0
10	push	local 1
11	add	
12	pop	local 0
13	push	local 1
14	push	local 0

Static

Index	Value
0	
1	
2	

Local

Index	Value
0	15
1	8
2	0

Argument

Index	Value
0	
1	

Breakpoint Panel

Variable Name	Value
local[1]	8
sp	271
line	13
currentFunction	Main.add

Stack

Call Stack

Function
Sys.init
Main.main
Main.add

RAM

Register	Value
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
Temp0	5
Temp1	6
Temp2	7
Temp3	8
Temp4	9
Temp5	10
Temp6	11
Temp7	12
R13	13
R14	14

Breakpoint reached

A powerful debugging tool!



# Breakpoints in Scripts

```
load myProg.vm,  
output-file myProg.out,  
output-list sp%D2.4.2  
           CurrentFunction%S1.15.1  
           Argument[0]%D3.6.3  
           RAM[256]%D2.6.2;  
  
breakpoint currentFunction Sys.init,  
  
set RAM[256] 15,  
set sp 257;  
  
repeat 3 {  
    vmStep,  
}  
output;  
  
while sp < 260 {  
    vmstep;  
}  
output;  
  
clear-breakpoints;  
  
// Etc.
```

- For systematic and replicable debugging, use scripts
- The first script commands usually load the `.vm` program and set up for the simulation
- The rest of the script may use various debugging-oriented commands:
  - Write variable values (output)
  - Repeated execution (while)
  - Set/clear Breakpoints
  - Etc. (see Appendix B.)

# End-note on Creating Virtual Worlds

---

"It's like building something where you don't have to order the cement. You can create a world of your own, your own environment, and never leave this room."

(Ken Thompson,  
1983 Turing Award lecture)



Ken Thompson (L) and Dennis Ritchie (R)