Virtual Machines

Pau Fernández

Universitat Politècnica de Catalunya (UPC)

What is a VM?

A virtual machine is a simulated computer that runs a special-purpose instruction set

Why do we need VMs?

We need Virtual Machines because...

- Implementing an AST interpreter is very inefficient.
- Surprisingly **easy to write** a Virtual Machine!
- **Much easier** to compile to a VM than to a real ISA.
- **Portability** to any architecture just by implementing the Virtual Machine on that architecture.
- We can express **special purpose algorithms** very succintly with a specially designed bytecode (Postscript, EVM, etc.)

(The best virtual machines do just-in-time compilation so they can run very fast.)

Bytecode

Binary instructions for Virtual Machines are called **bytecode**.

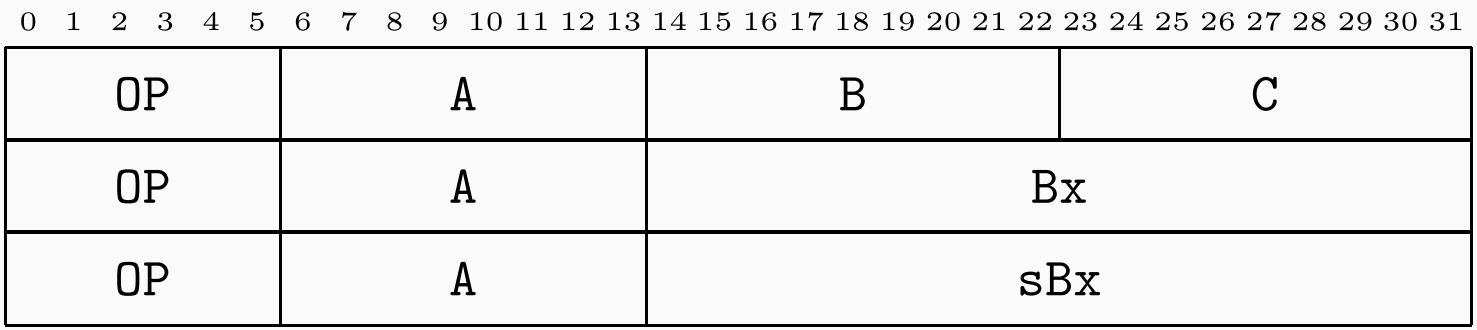
Bytecodes are composed of **opcodes** and **operands**.

Implementing a Virtual Machine entails designing the machine and its **instruction set**.

Instruction sets include, typically:

- Use of constants.
- Arithmetic operations.
- Comparisons (boolean expressions).
- Control flow: branches, jumps, calls.

Lua Bytecode




```
function max (a,b)
  local m = a           1 MOVE     2 0 0   ; R(2) = R(0)
  if b > a then         2 LT       0 0 1   ; R(0) < R(1) ?
    m = b               3 JMP      1       ; to 5 (4+1)
  end                   4 MOVE     2 1 0   ; R(2) = R(1)
  return m              5 RETURN   2 2 0   ; return R(2)
end                     6 RETURN   0 1 0   ; return
```

Stack vs Register

Stack Machines

Intermediate values are kept in a stack. (Reverse Polish Notation.)

Register Machines

Intermediate values are kept in registers (as do real machines).

Stack Machines

- Stack based virtual machines have very **compact bytecode**, because instructions are assumed to operate on the values at the top of the stack.
- Compact bytecode is good for the L1 cache, since it occupies little space.
- Stack machines are **easier** to write.
- It is easier to generate (compile) bytecode for stack machines.

Register Machines

Problem: stack machines often need extra instructions, because

- instructions *consume* the values of operands (DUP), or
- they are in a wrong order (SWAP).

Register machines may be faster because they can compress the computation better. (There is still an ongoing debate about this.)

What type do big projects use?

Stack machines

- EVM (Ethereum Virtual Machine)
- JVM (Java Virtual Machine)
- VES for CLI (Microsoft .NET)
- WebAssembly
- Adobe's Postscript
- CPython
- Ruby YARV

Register machines

- Lua
- Dalvik (Android)
- BEAM (Erlang)
- Parrot (Perl 6)

Our Stack Machine

Bytecodes are 1 byte (256 possible instructions).

All instructions assume a **value stack**.

Addition:

- Take 2 values from the stack.
- Add them.
- Put the result value back on the stack.

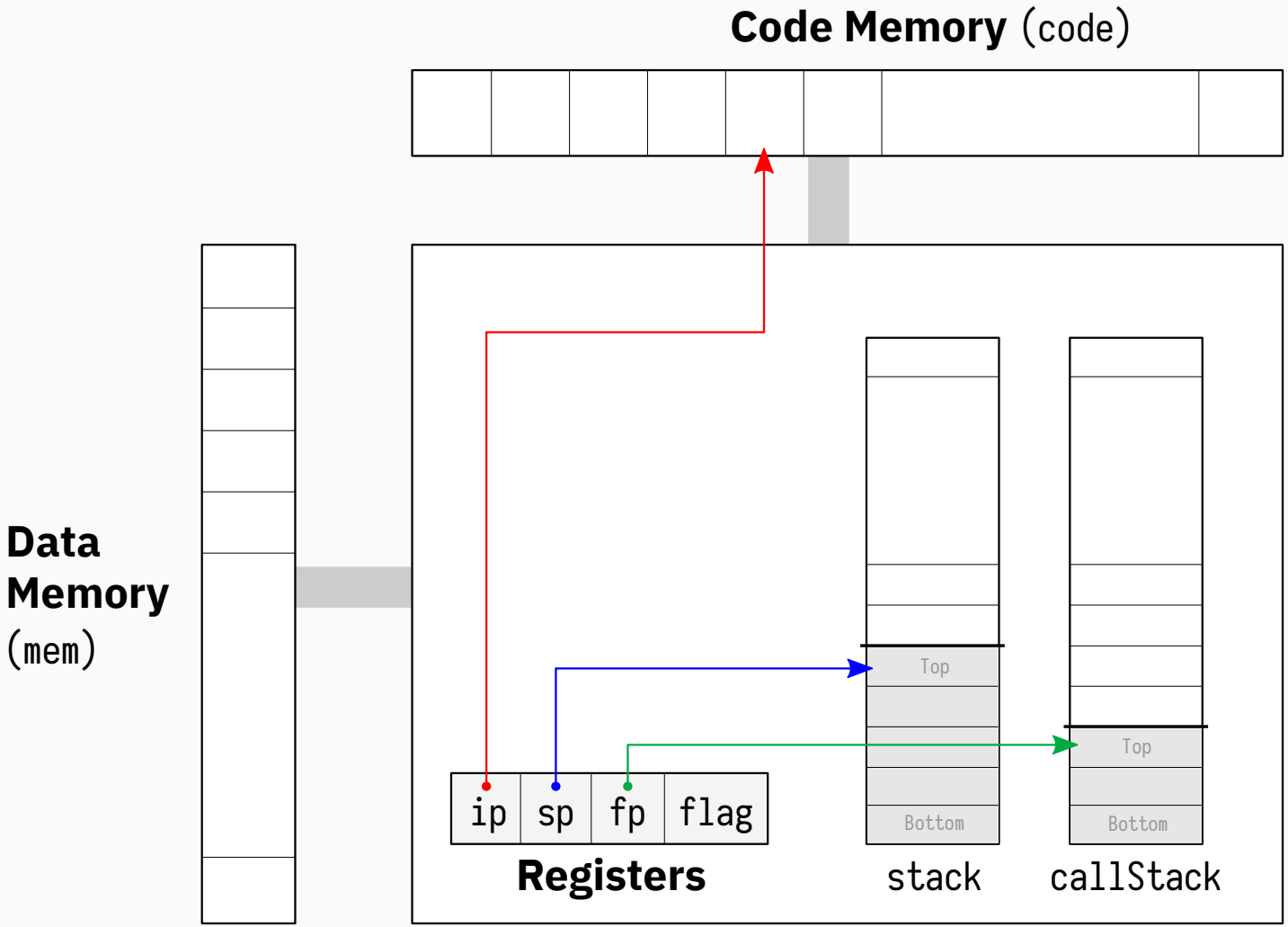
Instructions with operands

Most instructions **don't have operands** (all binary operations: ADD, SUB, LT, GT, ...).

But some of the instructions need one or two operands:

- Pushing constants on the stack.
- Branching, which needs the branching address.
- Calling a subroutine, which also needs the subroutine address.

Our Virtual Machine Structure



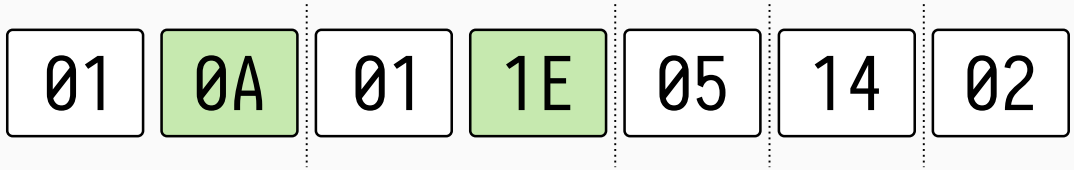
Our Instruction Set

00	HALT	Halt (stop) the machine
01	PUSH *val*	Pushes a value on to the stack
02	POP	Removes the top element from the stack
03	DUP	Duplicates the value at the top of the stack
05	ADD	Addition
06	SUB	Subtraction
07	MUL	Multiplication
08	INC	Increment by 1
10	LOAD *addr*	Load value from memory
11	STORE *addr*	Store top value to memory at *addr*
15	CALL *addr*	Call a subroutine
16	RET	Return from a subroutine
20	LT	Less-Than
21	EQ	Equals
22	BR *addr*	Branch (Jump)
23	BRF *addr*	Branch if ”flag” is `false`
24	BRT *addr*	Branch if ”flag” is `true`

...

Example Bytecode Execution

Bytecode



Bytecode	Mnemonics	Stack
01 0A	PUSH 10	10
01 1E	PUSH 30	10 30
05	ADD	40
14	PR	40
02	POP	Stack Bottom

Exercises!

- **PUSH0, PUSH1**: Add two new instructions to the virtual machine: one that pushes the value 0, and another for the value 1. Rewrite the programs you have that can use them. (These instructions can often save us some bytes!)
- **Hello World**: Write a program for this VM which shows ”Hello!”: 1) Put the string in memory (0-terminated); 2) Write a loop to iterate through the memory positions of every character and print them. If you need new instructions, just implement them.
- **Is Prime**: Write a subroutine and a program to check if a number is prime. Add any instructions you need that are general enough that you could use them in other programs.
- **Modular Arithmetic**: Modify the virtual machine to have a new register called ”m”. Write new arithmetic instructions