



# Compilers

---

## Stack Machines

- Only storage is a stack
- An instruction  $r = F(a_1, \dots, a_n)$ :
  - Pops  $n$  operands from the stack
  - Computes the operation  $F$  using the operands
  - Pushes the result  $r$  on the stack

# Stack Machines

- Consider two instructions
  - **push i** - push integer **i** on the stack
  - **add** - add two integers
  - A program:

**push 7**

**push 5**

**add**

- Stack machines are a very simple machine model
  - Leads to a simple, small compiler
  - But not necessarily one that produces very fast code

- Location of the operands/result is not explicitly stated
  - Always the top of the stack
- In contrast to a *register machine*
  - `add` instead of `add r1, r2, r3`
  - More compact programs
- One reason that Java bytecode uses stack evaluation

- There is an intermediate point between a pure stack machine and a pure register machine
- *An  $n$ -register stack machine*
  - Conceptually, keep the top  $n$  locations of the pure stack machine's stack in registers
- Consider a 1-register stack machine
  - The register is called the *accumulator*

- In a pure stack machine
  - An **add** does 3 memory operations
  - Two reads and one write to the stack
- In a 1-register stack machine the **add** does
$$\text{acc} \leftarrow \text{acc} + \text{top\_of\_stack}$$



- Consider an expression  $op(e_1, \dots, e_n)$ 
  - Note  $e_1, \dots, e_n$  are subexpressions
- For each  $e_i$  ( $0 < i < n$ )
  - Compute  $e_i$
  - Push result on the stack
- Pop  $n-1$  values from the stack, compute  $op$
- Store result in the accumulator

# Stack Machines

After evaluating an expression **e**, the accumulator holds the value of **e** and the stack is unchanged.

*Expression evaluation preserves the stack.*

Code	Acc	Stack
$\text{acc} \leftarrow 3$	3	$\langle \text{init} \rangle$
push acc	3	3, $\langle \text{init} \rangle$
$\text{acc} \leftarrow 7$	7	3, $\langle \text{init} \rangle$
push acc	7	7, 3, $\langle \text{init} \rangle$
$\text{acc} \leftarrow 5$	5	7, 3, $\langle \text{init} \rangle$
$\text{acc} \leftarrow \text{acc} + \text{top\_of\_stack}$	12	7, 3, $\langle \text{init} \rangle$
pop	12	3, $\langle \text{init} \rangle$
$\text{acc} \leftarrow \text{acc} + \text{top\_of\_stack}$	15	3, $\langle \text{init} \rangle$
pop	15	$\langle \text{init} \rangle$

## Stack Machines

Given the current state of the stack and accumulator, what is the next line of code to generate for the code fragment

$(2 * 3) + 5$ ?

Current:

Acc : 5

Stack: 6,<init>

☐ push acc

☐ pop

☐  $\text{acc} \leftarrow 6$

☐  $\text{acc} \leftarrow \text{acc} + \text{top\_of\_stack}$