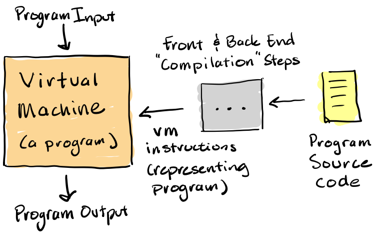
Lecture 20:

- Quiz 5
- VMs and the Instruction Set

Announcements:

- HW-4 out
- Proj. Part 1 due

Virtual Machines (VMs) for PL Interpretation


Program Input
Virtual Machine (a program)
Program Output
vm instructions (representing program)
Front & Back End "Compilation" Steps
...
Program Source code

VM implements an **abstract (computing) machine**

- similar to computer hardware (but in software) ...
- like a computer, consists of memory, instruction set, etc.
- assembly-like instructions ... load, store, add, jump, etc.

In a **bytecode** VM ... smaller instructions, easy to parse

- encodes instructions in binary as a sequence of bytes (e.g., `.class` files)
- e.g., `ADD 3 4` might be encoded for the VM as $\underbrace{0\ 1\ 1\ 0}_{\text{"opcode"}}\underbrace{0\ 0\ 1\ 1}_{\text{"3"}}\underbrace{0\ 1\ 0\ 0}_{\text{"4"}}$

MyPL VM for HW-5 and HW-6

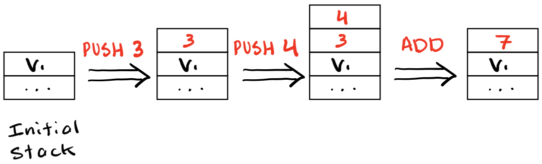
Loosely based on JVM architecture (stack machine, stack frames)

- Short-cuts plus API calls instead of using bytecode encoding/decoding

(1) Data Types/Values

- Java types to represent values … assumes programs are well typed

(2) Abstract Stack Machine … instead of registers, uses an operand stack



The VM components include: … more later

- operand stack (see above)
- memory for storing local variables … list of values
- struct and array heap storage … oid → {field:value}, oid → [value]
- function-call stack … i.e., stack of call “frames”

MyPL VM Instruction Set

(3) MyPL VM Instruction Set (high level) … see `OpCode.java`

Note: OP(A) says A is supplied directly to the OP instruction

- instructions take inputs directly and/or from the operand stack
- difference is what can be provided *statically* (directly)
- … versus *dynamically* to instruction

(a) Literals and variables

PUSH(A)	push value A onto the operand stack
POP()	pop value off of the stack (remove value)
STORE(A)	pop x, store x at memory address A (a list index)
LOAD(A)	fetch x at memory address A, push x on to operand stack