**Lecture 22:**

- Instruction Set (wrap up)

**Announcements:**

- HW-4 due date extended to Fri
- HW-5 out
- Proj Part 2 ...



## MyPL VM Instruction Set

**From Last Time ...**

| | |
|---|---|
| `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 |
| `ADD()` | pop $x$, pop $y$, push $(y + x)$ on to operand stack |
| `SUB()` | pop $x$, pop $y$, push $(y - x)$ on to operand stack |
| `MUL()` | pop $x$, pop $y$, push $(y \times x)$ on to operand stack |
| `DIV()` | pop $x$, pop $y$, push $(y \div x)$ on to operand stack |
| `AND()` | pop bool $x$, pop bool $y$, push $(y$ && $x)$ |
| `OR()` | pop bool $x$, pop bool $y$, push $(y$ \|\| $x)$ |
| `NOT()` | pop bool $x$, push $(!y)$ |
| `CMPLT()` | pop $x$, pop $y$, push $(y < x)$ |
| `CMPLE()` | pop $x$, pop $y$, push $(y \leq x)$ |
| `CMPEQ()` | pop $x$, pop $y$, push $(y$ == $x)$ |
| `CMPNE()` | pop $x$, pop $y$, push $(y$ != $x)$ |
| `JMP(`$A$`)` | jump to instruction $A$ (int index into instruction list) |
| `JMPF(`$A$`)` | pop $x$, if $x$ is false jump to instruction $A$ (int index) |

## MyPL VM Instruction Set

(f) Special instructions

| | |
|---|---|
| `DUP()` | pop $x$, push $x$, push $x$ |
| `NOP()` | no effect (a landing spot when jumping over code segments) |

(g) Functions (more details later)

| | |
|---|---|
| `CALL(`$A$`)` | calls function named $A$ |
| `RET()` | exit from function "returning" $x$ at top of stack |

(*) assume MyPL functions always return a value (add `null` if needed)



## MyPL VM Instruction Set

(h) Heap (more details later)

| | |
|---|---|
| `ALLOCS()` | allocate struct object in struct heap, push oid |
| `SETF(`$A$`)` | pop value $x$, pop oid $y$, in heap set `obj(`$y$`)[`$A$`] =` $x$ |
| `GETF(`$A$`)` | pop oid $x$, push `obj(`$x$`)[`$A$`]` on to operand stack |
| `ALLOCA()` | pop $x$, allocate array in array heap with $x$ `null` values, push oid |
| `SETI()` | pop value $x$, pop index $y$, pop oid $z$, set array `obj(`$z$`)[`$y$`] =` $x$ |
| `GETI()` | pop index $x$, pop oid $y$, push `obj(`$y$`)[`$x$`]` on to operand stack |

(*) where `obj(`$x$`)` means get (array or struct) object with oid $x$ from heap

## MyPL VM Instruction Set

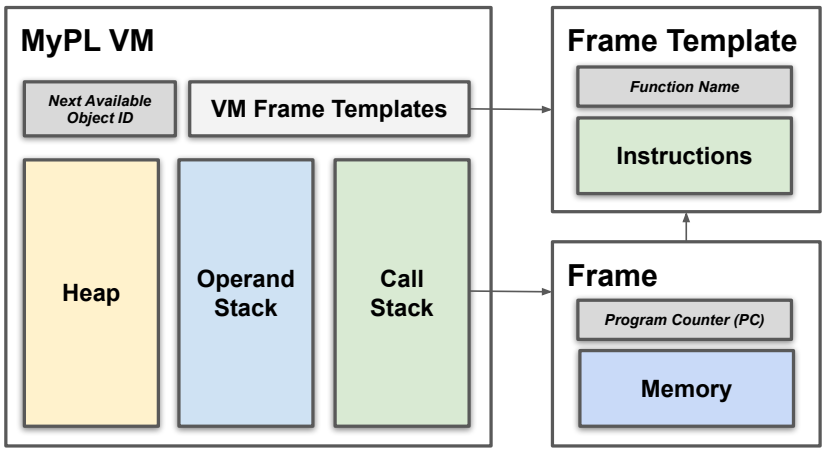
(i) Built-in functions

| | |
|---|---|
| `WRITE()` | pop $x$, print $x$ to stdout |
| `READ()` | read line $x$ from stdin |
| `LEN()` | pop $x$, push length of string or array |
| `GETC()` | pop string $x$, pop index $y$, push $x$`.charAt(`$y$`)` |
| `TOINT()` | pop value $x$, push $x$ as int |
| `TODBL()` | pop value $x$, push $x$ as double |
| `TOSTR()` | pop $x$, push $x$ as string |



## MyPL VM Architecture

**(4) Basic MyPL VM architecture**




- we separate into `VMFrameTemplate` and `VMFrame`
- each `VMFrame` can be thought of as an "instance" of the template
- a `VMFrame` holds a reference to its template (e.g., for instructions)