**Lecture 24:**

- MyPL Virtual Machine (cont)

**Announcements:**

- HW-4 out

## Instructions From Last Time:

| | |
|---|---|
| `PUSH(`$A$`)` | push argument $A$ onto the operand stack |
| `POP()` | pop value off of the stack |
| `STORE(`$A$`)` | pop $x$, store $x$ at memory address $A$ (a list index) |
| `LOAD(`$A$`)` | get $x$ at memory address $A$, push $x$ on to stack |
| `ADD()` | pop $x$, pop $y$, push $(y + x)$ on to stack |
| `SUB()` | pop $x$, pop $y$, push $(y - x)$ on to stack |
| `MUL()` | pop $x$, pop $y$, push $(y \times x)$ on to stack |
| `DIV()` | pop $x$, pop $y$, push $(y \div x)$ on to stack |
| `AND()` | pop bool $x$, pop bool $y$, push ($y$ `and` $x$) |
| `OR()` | pop bool $x$, pop bool $y$, push ($y$ `or` $x$) |
| `NOT()` | pop bool $x$, push (`not` $y$) |
| `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) |

## Simple example from last time: `while j < 3 { j = j + 1 } ...`

```
0:  LOAD(0)       # assume j stored in variables[0]
1:  PUSH(3)       # literal value for the comparison
2:  CMPLT()       # true if j < 3
3:  JMPF(9)       # if j >= 3, jump to instruction 9
4:  LOAD(0)       # get j again
5:  PUSH(1)       # for the literal value 1
6:  ADD()         # compute j + 1
7:  STORE(0)      # store result back into j
8:  JMP(0)        # go back to start of while
9:  ...           # continue on after while loop
```

(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 return a value (and we add a null value if needed)

(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

(i) Built-in functions

| | |
|---|---|
| `WRITE()` | pop $x$, write $x$ to stdout (via Python `print()`) |
| `READ()` | read line $x$ from stdin (via Python `input()`), push $x$ |
| `LEN()` | pop $x$, if `type(`$x$`) == str` push `len(`$x$`)`, else push `len(obj(`$x$`))` |
| `GETC()` | pop index $x$, pop string $y$, push $y[x]$ |
| `TOINT()` | pop value $x$, push `int(`$x$`)` |
| `TODBL()` | pop value $x$, push `float(`$x$`)` |
| `TOSTR()` | pop $x$, push `str(`$x$`)` |