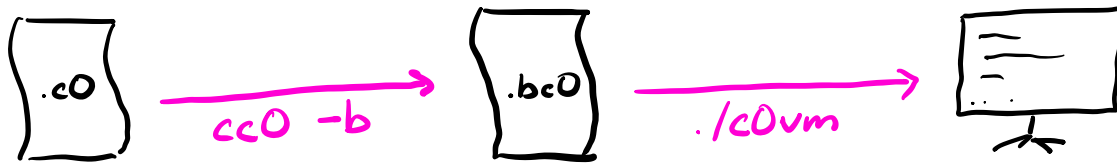


COVM



Overview

↪ most operations happen on the stack

- stack machine (like C1ac)
 - supplemented with local variable array
- compiler turns code into **bytecode** represented in an array
 - can move forwards and backwards on it
- COVM reads the bytecode instructions and does stuff
 - instruction opcodes take 1 byte, but some come with additional bytes of operands
 - bipush = 0x10, next byte tells what to push
 - ldc = 0x13, next 2 bytes tell what int pool index to get
- for control flow (goto, if_*), the offset is from the index of the actual instruction!
- function arguments are pre-loaded into its variable array at runtime, starting at index 0 and in order of how they appear

ex) translate this CO code into bytecode

```
int fun(int a, int b) {  
    int games = a + b;  
    return games * games;  
}
```

```
Arithmetic  
0x60 iadd      S, x:w32, y:w32 -> S, x+y:w32  
0x64 isub      S, x:w32, y:w32 -> S, x-y:w32  
0x68 imul      S, x:w32, y:w32 -> S, x*y:w32  
0x6C idiv      S, x:w32, y:w32 -> S, x/y:w32  
0x70 irem      S, x:w32, y:w32 -> S, x%y:w32  
0x7E iand      S, x:w32, y:w32 -> S, x&y:w32  
0x80 ior       S, x:w32, y:w32 -> S, x|y:w32  
0x82 ixor      S, x:w32, y:w32 -> S, x^y:w32  
0x78 ishl      S, x:w32, y:w32 -> S, x<<y:w32  
0x7A ishr      S, x:w32, y:w32 -> S, x>>y:w32
```

```
Local Variables  
0x15 vload <i>  S -> S, v          (v = V[i])  
0x36 vstore <i> S, v -> S          (V[i] = v)
```

#<fun>

— # of arguments
— # of local variables
— — code length

BO

return

functions n things

- **native** functions are CO library functions (printint, string-join, etc.)
- **static** functions are user defined functions
- call stack + frames are how we keep track of previous function states so we can pick up where we left off