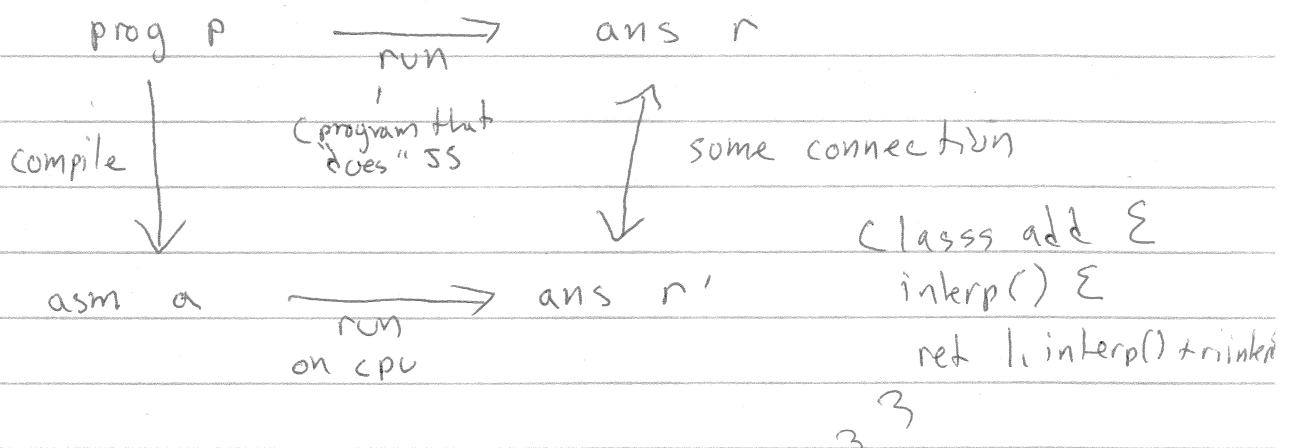
compiler : syntax of A / AST — tree (abstract syntax) ⇒ syntax of B

JS ⟶ x86

C ⟶ x86




```
interp ( p : prog ) =
  match p with
    | num n => n
    | add L R =>
        let lv = interp L in
        let rv = interp R in
        lv + rv
  end
```

---

exp = int (5, 6, 7) | (- exp) ← intrinsics
| (+ exp exp)
| var | (let ([var exp]) exp)

var = whatever (strings) | (read)

R1 = (program exp)

(+ 10 32) ⟶ 42 (- 10) ⟶ -10

(+ 10 (read)) ⟶ 10 20

(let ([x 10]) (+ x x)) ⟶ 20

(let ([x 10]) (let ([x (+ x 1)]) (+ x x))) ⟶ 22

1-2

Assembly

reg = rsp, rbp, rax, rbx, rcx, rdx, rsi, rdi, r8 – r15

arg = $int | %reg | int(reg)

prog = .globl main
main : INSTR+

INSTR = addq arg, arg (src, dest) — dest ← src + dest % 2^{64}

subq arg, arg

negq arg — dest ← negate dest

movq arg, arg (src, dest) — dest ← src

callq label
_read

0(rsp)
4(r8)

jayc → asm →(gas) j.o → ld → a.out
cc read.c → read.o → ld

pushq arg
popq arg
retq

(+10 32) ⇒

```
movq $10, %rax
addq $32, %rax
movq %rax, %rdi
callq print_int
retq
```

(+ 52 (- 10))

```
pushq %rbp
movq %rsp, %rbp
subq $16, %rsp

movq $10, -8(%rbp)
negq -8(%rbp)
movq $52, %rax
addq -8(%rbp), %rax

movq %rax, %rdi
callq print_int → addq $16, %rsp
popq %rbp
retq
```