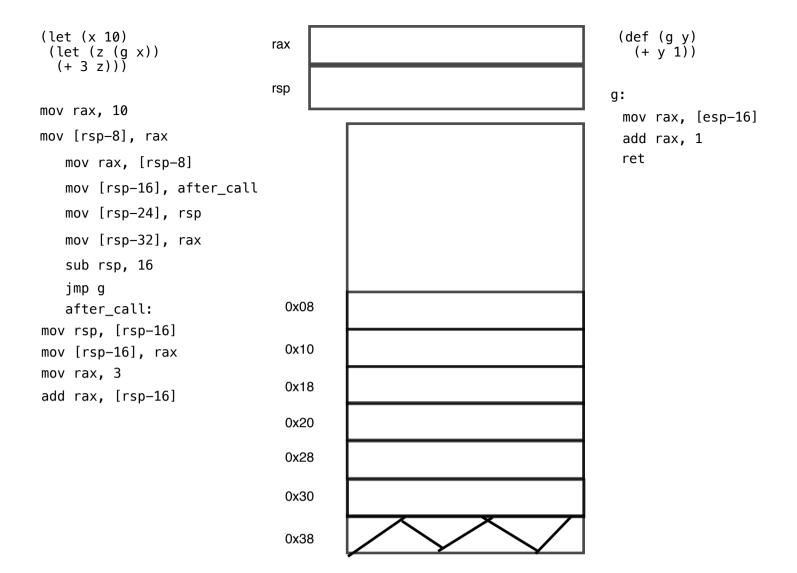
```
(+(-31)^2) \rightarrow (+(2)^2)
     (le+ (x 4) (+ x x)) → (+ 4 4) → 8
                                                                 (le+ (x 5)
                                             (le+ (x 5)
                          ( Set
                                                   X)
       X)
 (let (x (+12)) → (let (x 3 ) → (let (x 3 )
                                                        (let (y 4)
                              (let (y (+ x 1))
     (let (y (+ x 1))
                                                           (((r x +))
                                  (((r x +))
         (+ x y)))
~> (let (x 3
                                  (let (x 3 )
(let (Y 4
        (let (y 4)
            (+ 3 4)))
                                  (le+ (x -3)
(def (abs x)
                                     (if (< x 0) (* -1 x) x))
  (if (< x 0) (* -1 x) x))
(abs -3)
           type expr =
                                 - no types (for lecture/notes)
             I EApp of string * expr
           type def =
             | DFun of string * string * expr
           type prog = def list * expr
           let parse_def (sexp : Sexp.t) : def = ... parse ...
           let parse_program (sexps : Sexp.t list) : prog = ... parse ...
           let rec e_to_is (e : expr) (si : int) (env : tenv) (defs : def list) ; string list =
            ... other cases as before ...
           | EApp(f, arg) ->
                                                              INLINING!
             match find-def f defs with
                  1 Non - failwith "No def"
                                                             Poes not work injerval
                  I Some (DFun(rue, argnore, body))
                     e-to-is (ELer(argname, arg, body)) si en Jefs
```

let compile (program : Sexp.t list) : string =
 let (defs, body) = parse\_program program in
 let instrs = e\_to\_is body 1 [] defs in



One possible calling convention, but not the only one possible!

## Call setup:

- Move return address, then current rsp, then argument
- Always start at current si for return address, count up
- Subtract to point rsp at the return address

## Callee:

- Rely on (first) argument in [esp-16], so env starts with [(arg, 2)]
  - Start at a "higher" si=3 for any local vars
- Expect [rsp] to contain return pointer, use ret

## After the call:

- Rely on old rsp at [rsp-16] (a true constant)
- Expect answer to be in rax from callee

Release today or ton.

- Resubstiting older PAS - get back 50% of the autograded credit

- Read Campusuine post about exam