Calling convention for one argument:

Call setup (one arg version):
- 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

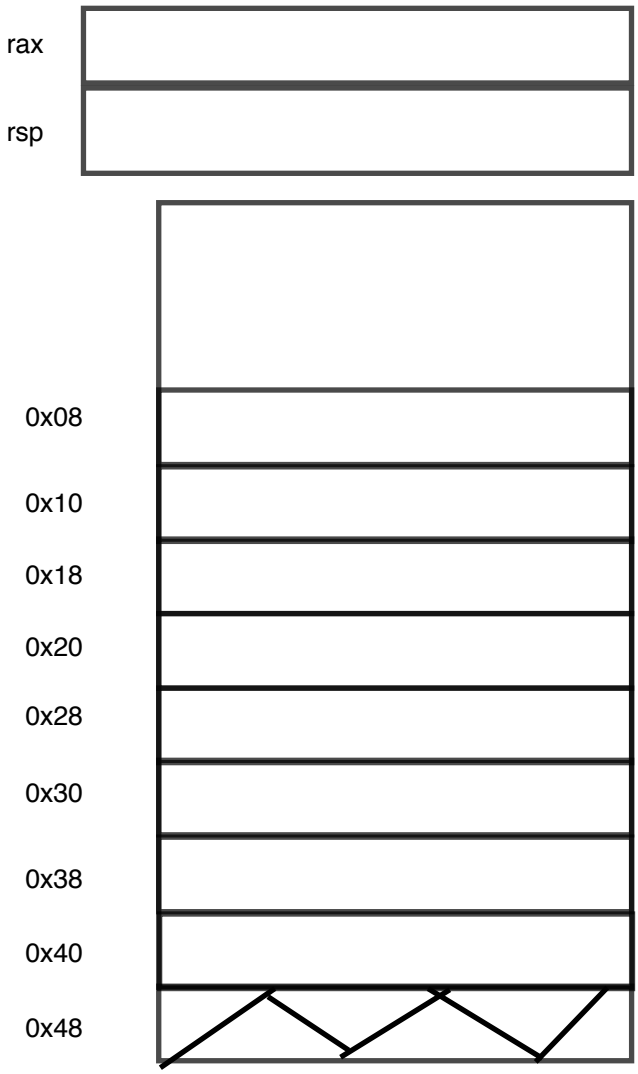
Callee (one arg version):
- Rely on (first) argument in [esp-16], so env starts with [(arg, 2)]
- Start at a “higher” si=3 for any local vars
- Expect [rsp] to contain return pointer, use ret

After the call (one arg version):
- Rely on old rsp at [rsp-16] (a true constant)
- Expect answer to be in rax from callee

What about two arguments?

```
(let (x 10)
 (let (z (g (+ x 1) (+ x 2)))
  (+ 3 z)))
```

```
   mov rax, 10
   mov [rsp-8], rax
```



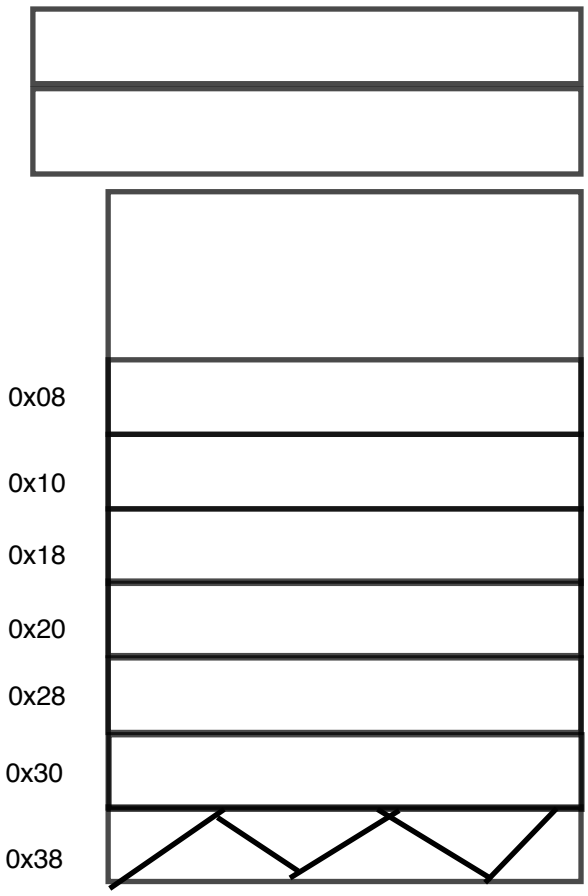

```
(def (g a b)
  (+ a b))

g:
```

```
(def (twosqrt3x x)
 (let (ans (do_sqrt (* x 3)))
  (* 2 ans)))
(twosqrt3x 4)
```

```
int do_sqrt(int val) {
  float asF = (float)val;
  return (int)(sqrt(asF));
}
```

```
  twosqrt3x:
     mov rax, [rsp-16]
     imul rax, 3
     sub rsp, 16
     mov rdi, rax
     call do_sqrt
0x900 mov rax, 4
     add rsp, 16
     mov [rsp-24], rax
     mov rax, 2
     imul rax, [rsp-24]
     ret
  our_code_starts_here:
     mov [rsp-8], after_call
     mov [rsp-16], rsp
     mov [rsp-24], rax
     sub rsp, 8
     jmp twosqrt3x
     after_call:
     ret
```




```
(def (fact n)
  (if (< n 2) 1
      (* n (fact (- n 1)))))
(fact 3)
```

```
(def (fact n sofar)
  (if (< n 2) sofar
      (fact (- n 1) (* n sofar))))
(fact 3 1)
```