```
(def (g y)
  (+ y 1))

(def (f x)
  (+ (g (+ x 2)) 3))

(def (main input)
  (f (+ input 4)))
```

fun name to call | argument value

fun name defined

function body

argument name

Later in quarter

```
type expr =
  | ENum of int
  | EBool of bool
  …
  | EDef of string * string * expr
  | EApp of expr * expr
```

Today/Tuesday

```
type expr =
  | ENum of int
  | EBool of bool
  …
  | EApp of string * expr

type def =
  | Def of string * string * expr

type prog =
  | Prog of def list * expr
```

Which representation do you want to implement first?

Some things to discuss:

- Compiling the new abstract syntax (getting its answer into EAX)
- How the environment works (a new kind of name)
- Are there programs we can represent with one but not the other?

```
type expr =
  | ENum of int
  | EBool of bool
  …
  | EApp of string * expr

type def =
  | Def of string * string * expr

type prog =
  | Prog of def list * expr
```

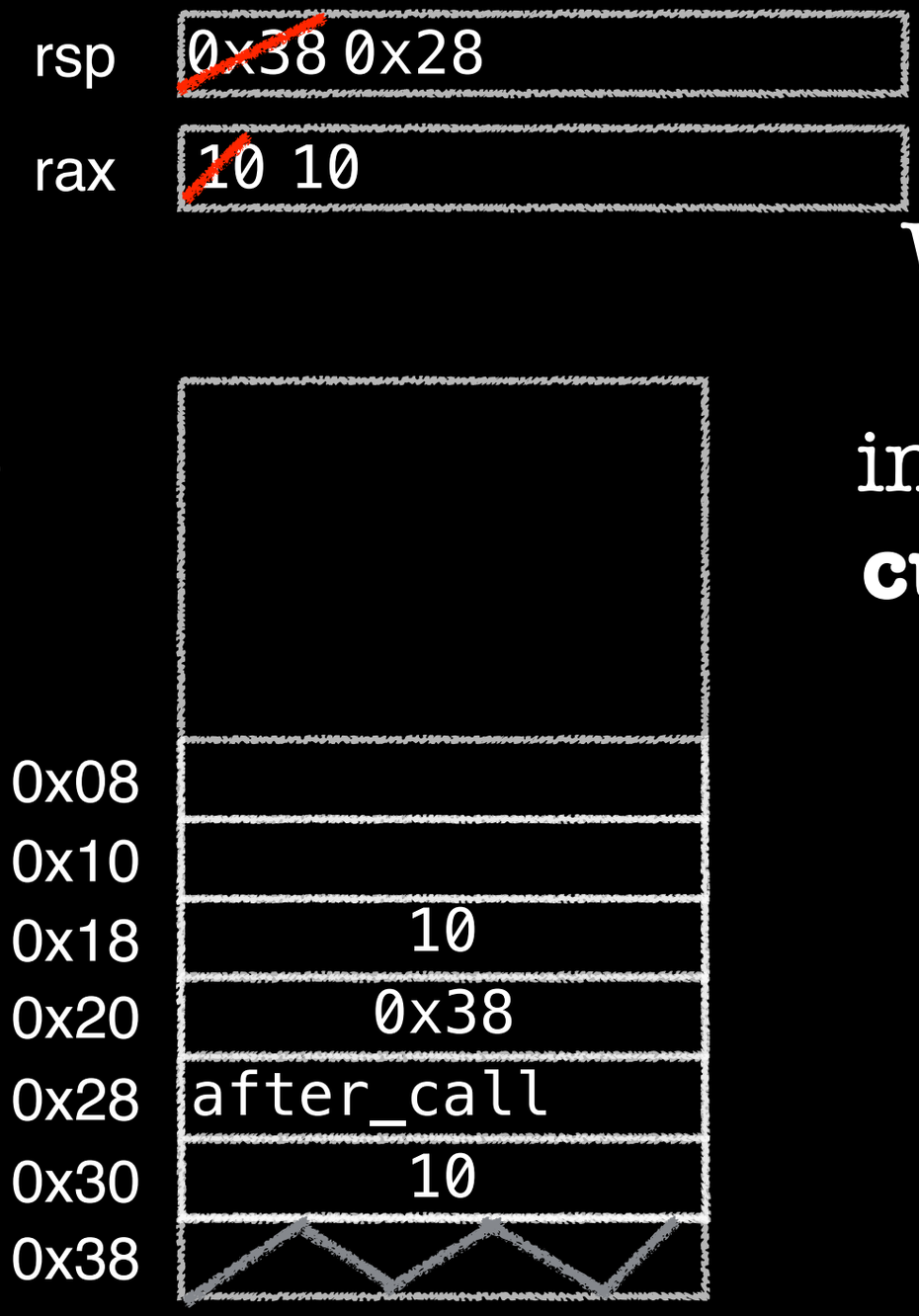
```
let rec compile_expr e si env =
  match e with
    …
    | EApp(fname, arg) -> …


let compile_def d … =
  …

let compile_prog p … =
  …
```

```
let rec compile_expr e si env =
  match e with
    …
    | EApp(fname, arg) ->
      INSTRUCTIONS FOR FUNCTION CALL
```

```
let compile_def d … =
  INSTRUCTIONS FOR FUNCTION BODY
```

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

```
mov rax, 10
mov [rsp-8], rax
  mov rax, [rsp-8]
  mov [rsp-16], after_call
  mov [rsp-24], rsp
  mov [rsp-32], rax
  sub rsp, 16
  jmp g
  after_call:
```




```
(def (g y)
  (+ y 1))
```

```
g:
```

Where is the argument y in terms of the **current value** of rsp?

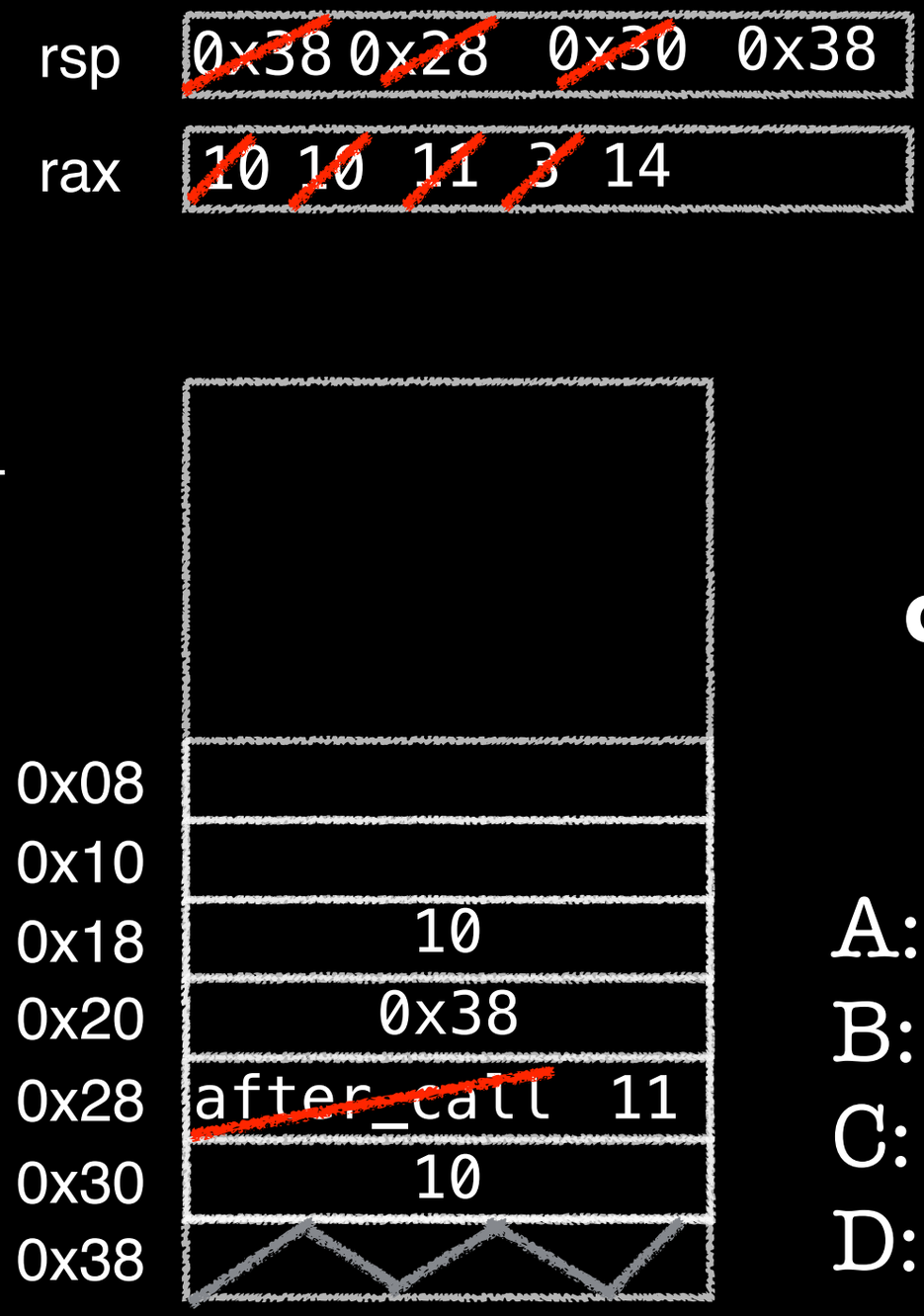
A: rsp
B: rsp−8
C: rsp−16
D: rsp+8
E: rsp−24

```
let rec compile_expr e si env =
  match e with
    …
    | EApp(fname, arg) ->
      INSTRUCTIONS FOR FUNCTION CALL
```

```
let compile_def d … =
  INSTRUCTIONS FOR FUNCTION BODY
```

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

```
mov rax, 10
mov [rsp-8], rax
 mov rax, [rsp-8]
 mov [rsp-16], after_call
 mov [rsp-24], rsp
 mov [rsp-32], rax
 sub rsp, 16
 jmp g
 after_call:
mov rsp, [rsp-16]
mov [rsp-16], rax
mov rax, 3
add rax, [rsp-16]
```




```
(def (g y)
  (+ y 1))
```

```
g:
mov rax, [rsp-16]
add rax, 1
ret
```

What is the **current value** in [rsp-16]?

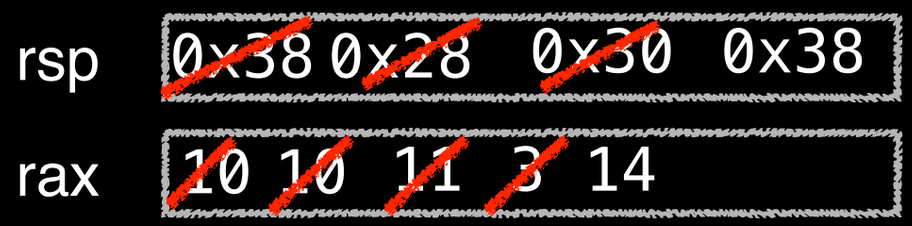
A: 10
B: 0x38
C: after_call
D: the other 10

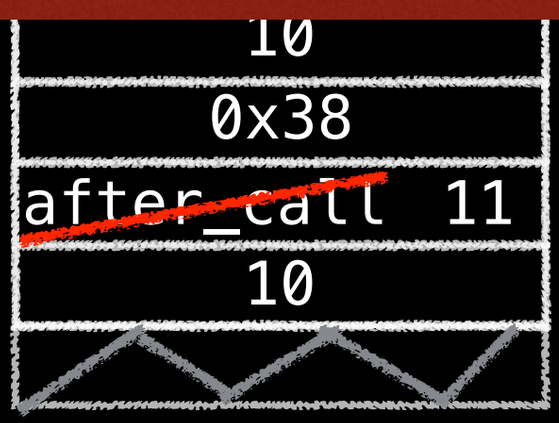
```
let rec compile_expr e si env =
  match e with
    …
    | EApp(fname, arg) ->
      INSTRUCTIONS FOR FUNCTION CALL
```

```
let compile_def d … =
  INSTRUCTIONS FOR FUNCTION BODY
```

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

```
mov rax, 10
mov [rsp-8], rax
 mov rax, [rsp-8]
 mov [rsp-16], after_call
 mov [rsp-24], rsp
 mov [rsp-32], rax
 sub rsp, 16
 jmp g
 after_call:
 mov rsp, [rsp-16]
 mov [rsp-16], rax
mov rax, 3
add rax, [rsp-16]
```






```
(def (g y)
  (+ y 1))
```

```
g:
mov rax, [rsp-16]
```

Call setup:

- Always these 3 values
- Always this order
- Always start at current si
- Always subtract to point rsp at the return address

A: 10
B: 0x38
C: after_call
D: the other 10

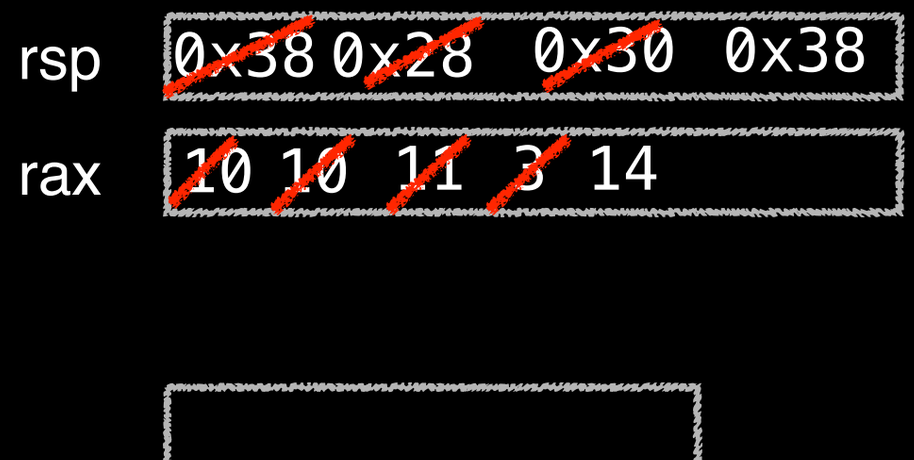
```
let rec compile_expr e si env =
  match e with
    …
    | EApp(fname, arg) ->
      INSTRUCTIONS FOR FUNCTION CALL
```

```
let compile_def d … =
  INSTRUCTIONS FOR FUNCTION BODY
```

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

```
(def (g y)
  (+ y 1))
```

```
mov rax, 10
mov [rsp-8], rax
  mov rax, [rsp-8]
  mov [rsp-16], after_call
  mov [rsp-2
  mov [rsp-3
  sub rsp, 1
  jmp g
  after_call
 mov rsp, [
 mov [rsp-16], rax
mov rax, 3
add rax, [rsp-16]
```




```
g:
mov rax, [rsp-16]
add rax, 1
ret
```

What is the

Callee has an easy job:

- 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

C: after_call

D: the other 10

```
let rec compile_expr e si env =
  match e with
    …
    | EApp(fname, arg) ->
      INSTRUCTIONS FOR FUNCTION CALL
```

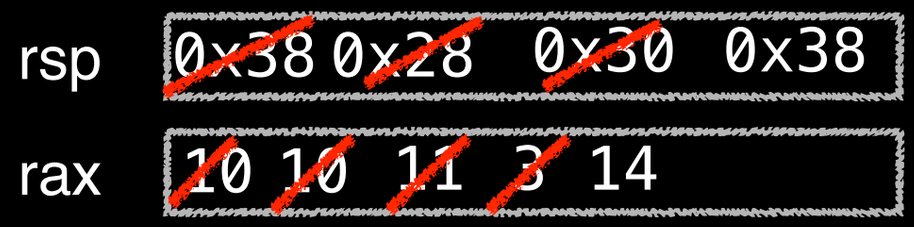
```
let compile_def d … =
  INSTRUCTIONS FOR FUNCTION BODY
```

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

```
(def (g y)
  (+ y 1))
```

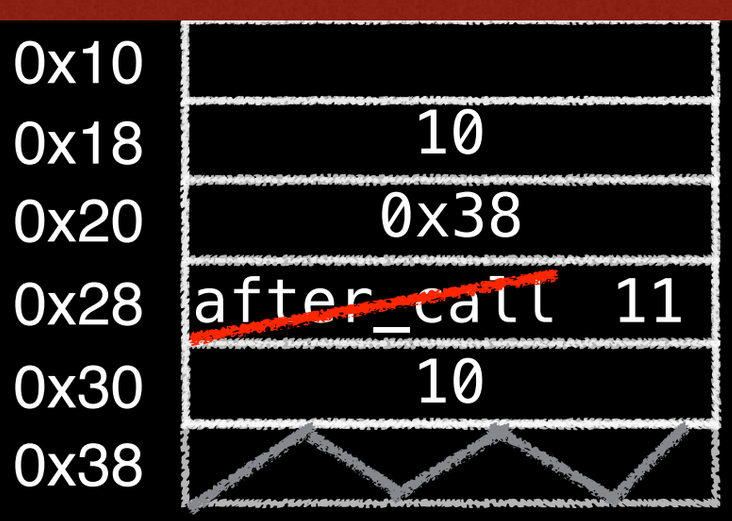
```
mov rax, 10
mov [rsp-8], rax
  mov rax, [rsp-8]
  mov [rsp-16], after_call
  mov [rsp-2
  mov [rsp-3
  sub rsp, 1
  jmp g
 after_call:
 mov rsp, [rsp-16]
  mov [rsp-16], rax
mov rax, 3
add rax, [rsp-16]
```

```
g:
mov rax, [rsp-16]
add rax, 1
ret
```



After the call:

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



A: 10
B: 0x38
C: after_call
D: the other 10

```
let rec compile_expr e si env =
  match e with
    …
    | EApp(fname, arg) ->
      INSTRUCTIONS FOR FUNCTION CALL
```

```
let compile_def d … =
  INSTRUCTIONS FOR FUNCTION BODY
```