Calling Convention for Diamondback

1. a)

The caller first stores the arguments required for the function on the stack. There is a helper function that compiles the expression list in EApp to get the arguments. The helper returns (instr', env', si', def\_env') so we get an updated stack index of up to where the arguments are stored. We can use the updated index minus the old index to find out the number of arguments applied. Then the caller is responsible for moving the return address onto the stack and storing the value of RSP on top of the return address. After that, the caller will move the arguments required for the function to be stored on top of the address of RSP, then the caller will move RSP to point at the return address of the function before calling the function. After the function is called, RSP is still pointing at the return address and we need to set that back. Due to our calling convention, the old value of RSP will always be stored at [rsp-16], so we call [IMov(Reg(RSP), (stackloc 2))].

b)

The callee is responsible for retrieving the arguments on the stack, which are already stored on top of the return addresses, and executing the function and then making sure that the answer to the function is stored in RAX. A label of the function name, [ILabel(name)], is also created to provide a place for the caller to jump to. Furthermore, the callee needs to call ret, which will cause the code to jump back to the return address.

c)

A possible improvement could be to improve the memory usage and efficiency of this calling convention. Due to the setup of the call, we must move the arguments above the return addresses, which means that we have to use double the memory usage to store the same number of arguments. This also means there is inefficiency in having to move the same arguments onto a different part of the stack.

1. a)

**Example 1: Fibonacci**

**Source**

(def fibonacci (n : Num) : Num

(if (< n 2) 1 (+ (fibonacci (- n 1)) (fibonacci (- n 2)))))

(fibonacci 3)

**[Generated assembly](#Fibonacci)**

**Summarized assembly**

The setup the caller does in Fibonacci is to store the return address and RSP on the stack first. Since the function only takes in one argument, we then move that argument to the top of the address. The last step of the setup is to point RSP at the return address. Then, the function gets called and the argument is evaluated to see if it is less than two. If the argument is less than 2, then we have reached the base case. If not, the argument is decremented and Fibonacci is recursively called.

**Output**

3

**Example 2: isprime**

**Source**

(def remainder (x : Num y : Num) : Num (if (< (- x y) 1) (if (== x y) 0 x) (remainder (- x y) y)))

(def isprime (y : Num) : Bool (let ((n 2)) (while (< n y) (if (== (remainder y n) 0) (set n 99999999) (set n (+ n 1)))) (if (== n 99999999) false true)))

(isprime 5)

**[Generated assembly](#prime)**

**Summarized assembly**

The setup for isprime is similar to Fibonacci because there is only one argument. Within isprime, we assign a variable n to start at 2 and increment it in a while loop, which is where the recursive call takes place. Within isprime, we store the return address and the address of RSP, then the two arguments required for remainder before we make the function call. Within remainder, we either return the remainder or set up another recursive call to remainder.

**Output**

true

**Example 3: Even Odd**

**Source**

(def even (n : Num) : Bool (if (== n 0) true (odd (- n 1)))) (def odd (n : Num) : Bool (if (== n 0) false (even (- n 1)))) (def test() : Bool (print (even 30)) (print (odd 30)) (print (even 57)) (print (odd 57))) (test)

**[Generated assembly](#evenodd)**

**Summarized assembly**

When we call even or odd with one argument, we put the return address and RSP address on the stack and then move the argument on top. Within even and odd, if the input is not 0, then the functions recursively call each other until they reach the base case of zero and return a Boolean. ß

**Output**

true\n\false\n\false\ntrue\ntrue

b)

Each of the functions listed above exhibit different types of recursion. Fibonacci is a function that calls itself in the function body, isprime calls another function (remainder) recursively and even/odd call each other recursively. These three functions test different forms of recursion and how well EApp runs.

**TESTS**

**input/fibonacci.boa**

(def fibonacci (n : Num) : Num

(if (< n 2) 1 (+ (fibonacci (- n 1)) (fibonacci (- n 2)))))

(fibonacci 3)

**input/remainder.boa**

(def remainder (x : Num y : Num) : Num

(if (< (- x y) 1) (if (== x y) 0 x) (remainder (- x y) y)))

(remainder 13 2)

**input/isprime.boa**

(def remainder (x : Num y : Num) : Num (if (< (- x y) 1) (if (== x y) 0 x) (remainder (- x y) y)))

(def isprime (y : Num) : Bool (let ((n 2)) (while (< n y) (if (== (remainder y n) 0) (set n 99999999) (set n (+ n 1)))) (if (== n 99999999) false true)))

(isprime 5)

**input/deepstack.boa**

(def func1 (n1 : Num n2 : Num b1 : Bool ) : Num (if b1 (set n1 (func2 n1 (\* n1 n2))) (set n1 (func2 n1 (+ n1 n2)))))

(def func2 (n1 : Num n2 : Num) : Num (while (< n1 n2) (set n1 (\* 2 n1)) (print n1)) (if (< (func3 n1) n2) n2 n1))

(def func3 (n1 : Num) : Num (set n1 (- n1 1)))

(func1 3 7 true)

**Assembly:**

**Fibonacci**

section .text

extern error

extern print

extern printPrint

global our\_code\_starts\_here

fibonacci:

mov rax, [rsp -16]

mov [rsp -24], rax

mov rax, 5

mov [rsp -32], rax

mov rax, [rsp -24]

cmp rax, [rsp -32]

jl near temp\_true\_branch\_19

mov rax, 0

jmp near temp\_end\_equals\_20

temp\_true\_branch\_19:

mov rax, 0x2

temp\_end\_equals\_20:

cmp rax, 0x2

jne near temp\_else\_branch\_23

mov rax, 3

jmp near temp\_end\_of\_if\_24

temp\_else\_branch\_23:

mov rax, [rsp -16]

mov [rsp -32], rax

mov rax, 3

sar rax, 1

shl rax, 1

mov [rsp -40], rax

mov rax, [rsp -32]

sub rax, [rsp -40]

jo near overflow

mov [rsp -32], rax

mov rbx, temp\_after\_call\_21

mov [rsp -40], rbx

mov [rsp -48], rsp

mov rax, [rsp -32]

mov [rsp -56], rax

sub rsp, 40

jmp near fibonacci

temp\_after\_call\_21:

mov rsp, [rsp -16]

mov [rsp -32], rax

mov [rsp -32], rax

mov rax, [rsp -16]

mov [rsp -40], rax

mov rax, 5

sar rax, 1

shl rax, 1

mov [rsp -48], rax

mov rax, [rsp -40]

sub rax, [rsp -48]

jo near overflow

mov [rsp -40], rax

mov rbx, temp\_after\_call\_22

mov [rsp -48], rbx

mov [rsp -56], rsp

mov rax, [rsp -40]

mov [rsp -64], rax

sub rsp, 48

jmp near fibonacci

temp\_after\_call\_22:

mov rsp, [rsp -16]

mov [rsp -40], rax

sar rax, 1

shl rax, 1

add rax, [rsp -32]

jo near overflow

temp\_end\_of\_if\_24:

ret

ret

expected\_num:

mov rdi, 11

push 0

call error

expected\_bool:

mov rdi, 21

push 0

call error

overflow:

mov rdi, 31

push 0

call error

our\_code\_starts\_here:

push rbx

mov [rsp - 8], rdi

mov rax, 7

mov [rsp -16], rax

mov rbx, temp\_after\_call\_25

mov [rsp -24], rbx

mov [rsp -32], rsp

mov rax, [rsp -16]

mov [rsp -40], rax

sub rsp, 24

jmp near fibonacci

temp\_after\_call\_25:

mov rsp, [rsp -16]

mov [rsp -16], rax

pop rbx

ret

**isprime**

section .text

extern error

extern print

extern printPrint

global our\_code\_starts\_here

remainder:

mov rax, [rsp -16]

mov [rsp -32], rax

mov rax, [rsp -24]

sar rax, 1

shl rax, 1

mov [rsp -40], rax

mov rax, [rsp -32]

sub rax, [rsp -40]

jo near overflow

mov [rsp -32], rax

mov rax, 3

mov [rsp -40], rax

mov rax, [rsp -32]

cmp rax, [rsp -40]

jl near temp\_true\_branch\_36

mov rax, 0

jmp near temp\_end\_equals\_37

temp\_true\_branch\_36:

mov rax, 0x2

temp\_end\_equals\_37:

cmp rax, 0x2

jne near temp\_else\_branch\_43

mov rax, [rsp -16]

mov [rsp -40], rax

mov rax, [rsp -24]

mov [rsp -48], rax

mov rax, [rsp -40]

cmp rax, [rsp -48]

jne near temp\_false\_branch\_38

mov rax, 0x2

jmp near temp\_end\_equals\_39

temp\_false\_branch\_38:

mov rax, 0

temp\_end\_equals\_39:

cmp rax, 0x2

jne near temp\_else\_branch\_40

mov rax, 1

jmp near temp\_end\_of\_if\_41

temp\_else\_branch\_40:

mov rax, [rsp -16]

temp\_end\_of\_if\_41:

jmp near temp\_end\_of\_if\_44

temp\_else\_branch\_43:

mov rax, [rsp -16]

mov [rsp -40], rax

mov rax, [rsp -24]

sar rax, 1

shl rax, 1

mov [rsp -48], rax

mov rax, [rsp -40]

sub rax, [rsp -48]

jo near overflow

mov [rsp -40], rax

mov rax, [rsp -24]

mov [rsp -48], rax

mov rbx, temp\_after\_call\_42

mov [rsp -56], rbx

mov [rsp -64], rsp

mov rax, [rsp -40]

mov [rsp -72], rax

mov rax, [rsp -48]

mov [rsp -80], rax

sub rsp, 56

jmp near remainder

temp\_after\_call\_42:

mov rsp, [rsp -16]

mov [rsp -40], rax

temp\_end\_of\_if\_44:

ret

isprime:

mov rax, 5

mov [rsp -24], rax

temp\_start\_while\_45:

mov rax, [rsp -24]

mov [rsp -32], rax

mov rax, [rsp -16]

mov [rsp -40], rax

mov rax, [rsp -32]

cmp rax, [rsp -40]

jl near temp\_true\_branch\_47

mov rax, 0

jmp near temp\_end\_equals\_48

temp\_true\_branch\_47:

mov rax, 0x2

temp\_end\_equals\_48:

cmp rax, 0x2

jne near temp\_end\_while\_46

mov rax, [rsp -16]

mov [rsp -32], rax

mov rax, [rsp -24]

mov [rsp -40], rax

mov rbx, temp\_after\_call\_49

mov [rsp -48], rbx

mov [rsp -56], rsp

mov rax, [rsp -32]

mov [rsp -64], rax

mov rax, [rsp -40]

mov [rsp -72], rax

sub rsp, 48

jmp near remainder

temp\_after\_call\_49:

mov rsp, [rsp -16]

mov [rsp -32], rax

mov [rsp -32], rax

mov rax, 1

mov [rsp -40], rax

mov rax, [rsp -32]

cmp rax, [rsp -40]

jne near temp\_false\_branch\_50

mov rax, 0x2

jmp near temp\_end\_equals\_51

temp\_false\_branch\_50:

mov rax, 0

temp\_end\_equals\_51:

cmp rax, 0x2

jne near temp\_else\_branch\_52

mov rax, 199999999

mov [rsp -24], rax

jmp near temp\_end\_of\_if\_53

temp\_else\_branch\_52:

mov rax, [rsp -24]

mov [rsp -40], rax

mov rax, 3

sar rax, 1

shl rax, 1

add rax, [rsp -40]

jo near overflow

mov [rsp -24], rax

temp\_end\_of\_if\_53:

jmp near temp\_start\_while\_45

temp\_end\_while\_46:

mov rax, 0

mov rax, [rsp -24]

mov [rsp -40], rax

mov rax, 199999999

mov [rsp -48], rax

mov rax, [rsp -40]

cmp rax, [rsp -48]

jne near temp\_false\_branch\_54

mov rax, 0x2

jmp near temp\_end\_equals\_55

temp\_false\_branch\_54:

mov rax, 0

temp\_end\_equals\_55:

cmp rax, 0x2

jne near temp\_else\_branch\_56

mov rax, 0

jmp near temp\_end\_of\_if\_57

temp\_else\_branch\_56:

mov rax, 0x2

temp\_end\_of\_if\_57:

ret

ret

expected\_num:

mov rdi, 11

push 0

call error

expected\_bool:

mov rdi, 21

push 0

call error

overflow:

mov rdi, 31

push 0

call error

our\_code\_starts\_here:

push rbx

mov [rsp - 8], rdi

mov rax, 7

mov [rsp -16], rax

mov rbx, temp\_after\_call\_58

mov [rsp -24], rbx

mov [rsp -32], rsp

mov rax, [rsp -16]

mov [rsp -40], rax

sub rsp, 24

jmp near isprime

temp\_after\_call\_58:

mov rsp, [rsp -16]

mov [rsp -16], rax

pop rbx

ret

**EvenOdd**

section .text

extern error

extern print

extern printPrint

global our\_code\_starts\_here

even:

mov rax, [rsp -16]

mov [rsp -24], rax

mov rax, 1

mov [rsp -32], rax

mov rax, [rsp -24]

cmp rax, [rsp -32]

jne near temp\_false\_branch\_45

mov rax, 0x2

jmp near temp\_end\_equals\_46

temp\_false\_branch\_45:

mov rax, 0

temp\_end\_equals\_46:

cmp rax, 0x2

jne near temp\_else\_branch\_48

mov rax, 0x2

jmp near temp\_end\_of\_if\_49

temp\_else\_branch\_48:

mov rax, [rsp -16]

mov [rsp -32], rax

mov rax, 3

sar rax, 1

shl rax, 1

mov [rsp -40], rax

mov rax, [rsp -32]

sub rax, [rsp -40]

jo near overflow

mov [rsp -32], rax

mov rbx, temp\_after\_call\_47

mov [rsp -40], rbx

mov [rsp -48], rsp

mov rax, [rsp -32]

mov [rsp -56], rax

sub rsp, 40

jmp near odd

temp\_after\_call\_47:

mov rsp, [rsp -16]

mov [rsp -32], rax

temp\_end\_of\_if\_49:

ret

odd:

mov rax, [rsp -16]

mov [rsp -24], rax

mov rax, 1

mov [rsp -32], rax

mov rax, [rsp -24]

cmp rax, [rsp -32]

jne near temp\_false\_branch\_50

mov rax, 0x2

jmp near temp\_end\_equals\_51

temp\_false\_branch\_50:

mov rax, 0

temp\_end\_equals\_51:

cmp rax, 0x2

jne near temp\_else\_branch\_53

mov rax, 0

jmp near temp\_end\_of\_if\_54

temp\_else\_branch\_53:

mov rax, [rsp -16]

mov [rsp -32], rax

mov rax, 3

sar rax, 1

shl rax, 1

mov [rsp -40], rax

mov rax, [rsp -32]

sub rax, [rsp -40]

jo near overflow

mov [rsp -32], rax

mov rbx, temp\_after\_call\_52

mov [rsp -40], rbx

mov [rsp -48], rsp

mov rax, [rsp -32]

mov [rsp -56], rax

sub rsp, 40

jmp near even

temp\_after\_call\_52:

mov rsp, [rsp -16]

mov [rsp -32], rax

temp\_end\_of\_if\_54:

ret

test:

mov rax, 61

mov [rsp -16], rax

mov rbx, temp\_after\_call\_55

mov [rsp -24], rbx

mov [rsp -32], rsp

mov rax, [rsp -16]

mov [rsp -40], rax

sub rsp, 24

jmp near even

temp\_after\_call\_55:

mov rsp, [rsp -16]

mov [rsp -16], rax

mov rdi, rax

sub rsp, 16

call print

add rsp, 16

mov rax, 61

mov [rsp -24], rax

mov rbx, temp\_after\_call\_56

mov [rsp -32], rbx

mov [rsp -40], rsp

mov rax, [rsp -24]

mov [rsp -48], rax

sub rsp, 32

jmp near odd

temp\_after\_call\_56:

mov rsp, [rsp -16]

mov [rsp -24], rax

mov rdi, rax

sub rsp, 16

call print

add rsp, 16

mov rax, 115

mov [rsp -32], rax

mov rbx, temp\_after\_call\_57

mov [rsp -40], rbx

mov [rsp -48], rsp

mov rax, [rsp -32]

mov [rsp -56], rax

sub rsp, 40

jmp near even

temp\_after\_call\_57:

mov rsp, [rsp -16]

mov [rsp -32], rax

mov rdi, rax

sub rsp, 32

call print

add rsp, 32

mov rax, 115

mov [rsp -40], rax

mov rbx, temp\_after\_call\_58

mov [rsp -48], rbx

mov [rsp -56], rsp

mov rax, [rsp -40]

mov [rsp -64], rax

sub rsp, 48

jmp near odd

temp\_after\_call\_58:

mov rsp, [rsp -16]

mov [rsp -40], rax

mov rdi, rax

sub rsp, 32

call print

add rsp, 32

ret

ret

expected\_num:

mov rdi, 11

push 0

call error

expected\_bool:

mov rdi, 21

push 0

call error

overflow:

mov rdi, 31

push 0

call error

our\_code\_starts\_here:

push rbx

mov [rsp - 8], rdi

mov rbx, temp\_after\_call\_59

mov [rsp -16], rbx

mov [rsp -24], rsp

sub rsp, 16

jmp near test

temp\_after\_call\_59:

mov rsp, [rsp -16]

mov [rsp -16], rax

pop rbx

ret