-1)

select : C ⇒ X (w/ vars and break some X86 rules)

$select_p$ (program ; [BODY ⇒ t]) = program : [BODY ⇒ (block ∅ ($select_t$ t))]

$select_t$ (return a) := [(movq ($select_a$ a) RAX) ; (jmp END)]

$selet_t$ (seq s t) := $select_s$ s ++ $select_t$ t

$select_a$ (num n) = (num n)
(var x) = (var x)

$select_s$ (set! x e) := ~~X movq (f e) x~~
$select_e$ x e
(selecta x)

$select_e$ dst a := [movq ($select_a$ a) dst]
$select_e$ dst (read) := [callq _read-int ; movq RAX dst]
(- a) := [movq ($select_a$ a) dst ; negq dst]
(+ a_L a_R) := [movq ($select_a$ a_R) dst ; addq ($select_a$ a_L) dst]

$$p \xrightarrow{uni} r' \xrightarrow{opt} r'' \xrightarrow{rco} r''' \xrightarrow{econ} C \xrightarrow{select} X \xrightarrow{assign} X \xrightarrow{patch} X$$

w/vars w/o vars

assign-holmes : X → X "register allocation"

assign (program ip [BODY ⇒ (block ib IS)])

$\overrightarrow{\text{local-vars}} = (x_1 \cdots x_n)$

$VC = \overbrace{8 \times (n \text{ or } n+1)}^{\text{must be div by 16}}$

(program (ip / local-vars) [
BEGIN ⇒ (block ∅ [pushq RBP ; movq RSP RBP ; subq VC , RSP ; jmp BODY])
END ⇒ (block ∅ [addq VC, RSP ; popq RBP ; retq]
BODY ⇒ (block ib (assign σ IS))]])

$\sigma = [X_1 \Rightarrow \%RSP(8 \times 1) , \ldots , X_n \Rightarrow \%RSP(8 \times n)]$

4-2/

```
assign_is σ []     = []
           (i:is)  = (assign σ i) : (assign σ is)

assign_i σ (addq aL,aR) = addq (assign σ aL) (assign σ aR)
           (negq aR)    = negq (assign σ aR)
           (movq aS,aD) = movq (      aS) (      aD)
           jmp LAB      = jmp LAB
           (callq LAB)  = callq LAB
assign_a σ (num n)      = (num n)
           (var x)      = σ(x)
```

patch recurs like assign   TMP = RAX

```
patch (addq R1(O1), R2(O2)) = [ movq R1(O1), TMP-REG
                                addq TMP-REG, R2(O2) ]
      (movq R1(O1), R2(O2)) = [ movq R1(O1), TMP
                                movq TMP, R2(O2) ]
      i                     = [i]

patch (i : is) = patch i ++ patch is
```

runtime.c   int64_t

```
int read_int() { int x; scanf("%d", &x); return x; }
void print_int(int x) { printf("%d", x); return 0; }
```

main : X → X
ans is in RAX   ans is printed & can be called

```
main (program ; blks)
  = (program ; blks +
      [_main => (block σ
         [ callq BEGIN;
           movq RAX, RDI;
           callq _print_int;
           retq ]])
```

.. print x86 to x.s ...

```
$ cc runtime.c x.s -o x.bin
$ ./x.bin
```

... read output ... turn into number...

... compare w/ expected ...