3-1

$R_0 \xrightarrow{\text{uniquify}} R_0 \xrightarrow{\text{flatten}} C_0 \xrightarrow{\text{select}} X_0^* \xrightarrow{\text{assign}} X_0 \xrightarrow{\text{fix}} X_0$

$R_0.e$ = int | (read) | (- e) | (+ e e) | var | (let *var e e)

p = (program e)

$C_0.p$ = (program (var*) stmt* arg) = <var*, stmt*, arg>

stmt = (:= var exp)

exp = arg | (read) | (- arg) | (+ arg arg)

arg = int | var

flatten : $R_0.p \Rightarrow C_0.p$

flattene : $R_0.e \Rightarrow C_0.p$

F(int) = <∅, , int>

F(- e) =

~~X <{nv}, nv := (- e), nv>~~

let <vs, ss, earg> = F(e)

<nv ++ vs, ss ++ (:= nv (- earg)), nv>

F(- (- 8)) = <∅, , 8>

<nv1, (:= nv (- 8)), nv>

<nv1 nv2, (:= nv1 (- 8)) (:=nv2 (-nv1)), nv2>

F(read) = <nv1, (rv := (read)), rv>

F(+ e1 e2) = <vs1, ss1, ea1> = F(e1)

<vs2, ss2, ea2> = F(e2)

<vs1 ++ vs2 ++ av, ss1 ++ ss2 ++ (av := (+ ea1 ea2)), av>

F(var) = <{var}, , var>

F(let x e1 e2) = < vs1 ++ vs2 ++ {x},

ss1 ++ [redacted] (:= x ea1) ++ ss2,

ea2 >

3-2/

X_0 = assembly

$X_0.p$ = instr +

$X_0.instr$ = (addq arg arg)
(movq arg arg)
(callq label)
(negq arg)
(retq arg)
(pushq arg)
(popq arg)
(subq arg arg)

arg = int | reg | (reg, offset)

X_0^* = X_0 but arg = int | var | reg

program = var*, $X_0.p$

select : $C_0.p \Rightarrow X_0^*.p$

S(<vs, ss, a>) = <vs, SS(ss) ++ (retq a)>

SS(mt) = mt

SS(stmt :: more) = S1(stmt) ++ SS(more)

S1 : $C_0.stmt \Rightarrow X_0^*.$instr*

S1(x := e) =

match on e

arg ⇒ (movq arg x)

read ⇒ (callq _read)
(movq rax x)

(- arg) ⇒ (movq arg x)
(negq x)

(+ arg_1 arg_2) ⇒ (movq arg_1 x)
(addq arg_2 x)

3-3/

assign : $X_0^x \Rightarrow X_0$

$\langle vars^x , instr+ \rangle \longrightarrow instr+$

mention variables

$X_0^*.arg = int \mid var \mid reg$

$X_0.arg = int \mid reg \mid (reg, offset)$

assign (< vars , instrs >) =

(subq (* 8 |vars|) rsp)

aintr ... (retq arg) = assign*(instrs) σ

aintr ... (movq arg rax)

σ(var) = offset

(addq (* 8 |vars|) rsp)

(retq rax)

assign* : (var ⇒ offset) instr → instr

assign*(σ, var v) = (rsp, −8 * σ(v))

3-4/ fix : $X_0 \Rightarrow X_0$

```
(let y (- 10)
     (+ y 52))
```

```
movq  10, x
negq  x
movq  x, y
addq  52, y
retq  y
```

assign ⇒

```
movq  10, rsp(-8)
negq  rsp(-8)
movq  rsp(-8), rsp(-16)      ← illegal
addq  52, rsp(-16)
movq  rsp(-16), rax
retq  rax
```

Cannot two memory references in one instruction

(movq a1 a2) if a1 = rsp(n)
(addq a1 a2) a2 = rsp(m)

0

src dest

(0 rsp(n) rsp(m))

⇒ (movq rsp(n) rax)
(0 rax rsp(m))