7-2

flatten : $R_2 \Rightarrow C_2$

$e = (+\ e\ e)$
$(if\ e\ e\ e)$

$a = num \mid var$
$e = (+\ a\ a) \mid (-\ a)$
$s = (ret\ a) \mid (set!\ x\ e)$
$p = s \ldots$

#t/#f → a
and/or/not/cmp → e
$if\ e^{R}_{e}$ → $if \in C_S$ ← US
↳ $x, y, z \in C_S$ ← patch step

$S = \ldots \mid (if\ a\ s\ s)$ (cmp a a)
not ret

flatten ( if $e_1$ $e_2$ $e_3$ ) = (vs, stmts ..., ret a)
$(vs_1, st_1, r_1)$ = flatten $(e_1)$ // cond
$(vs_2, st_2, r_2)$ = flatten $(e_2)$ // true
$(vs_3, st_3, r_3)$ = flatten $(e_3)$ // false
< $vs = \{ans\} \cup vs_{1,2,3}$
$st = st_1$ ; (if (eq? $r_1$ #t) $[st_2\ ans \leftarrow r_2]$ $[st_3\ ans \leftarrow r_3]$)
$r = ans$
>

optimize with
flattencmp = (vs, st..., (cmp a a))
fc(#t) = (∅, , (eq? #t #t))
fc(#f) = (∅, , (eq? #f #t))
fc((< $e_1$ $e_2$)) = ($vs_1 \cup vs_2$, $st_1 \ldots st_2 \ldots$,
↳ calls flatten (< $a_1$ $a_2$))
$(vs_1, st_1, a_1) = f(e_1)$
$(vs_2, st_2, a_2) = f(e_2)$

T(Γ, (if e #t 1))
= type
(int ∪ bool)
⇒ types as sets and predictions

(int ∩ bool) ⇒ types as "what you can do"
⇒ OO

⇒ more complicated if
(if (number? x)
(+ 1 x)
(strlen x))
x : (int ∪ str)

T : Γ × e → Ty : int ⇒ Racket
× Props that are true if e is true
× Props that are true if e is false

9-1 asm, select, liveness, interference, assign-homes, patch, optimizations

(+ x y) => addq y, x (deal with D ← D+S)

#t => $1 #f => $0

(not x) => notq x ~ 0001 => 1110

1 => 0

0 => 1

=> xorq x, $1

or subq $1, x

| Input | arg 1 | arg 0 |
|---|---|---|
| 1 | 1 | 0 |
| 0 | 0 | 0 |

andq x, ⌣

| xor | 1 | 0 |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |

D ← D-S

---

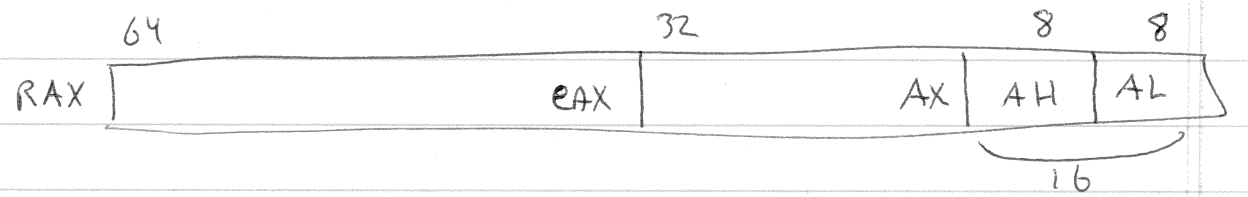
(< 3 4) => cmpq $3, $4 sets FLAGS

↙ branch ↘ read the flags

je – jump if eq is set

jle – jump if ≤ is set

=> sete dst




unsigned – movzbq 0...0 10010000 → 128+16 = 144

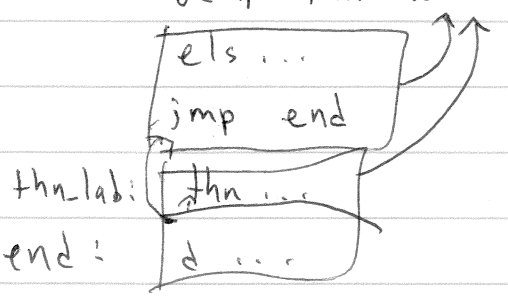
signed – movsbq ~~0...0~~ 1...1 (1)1111110 → −2

```
label:  movq
        addq
        cmpq
        je label
        movq
```

(if (cmp a1 a2) thn els) d...

=>

```
cmpq a1 a2
jcmp thn-lab:
```

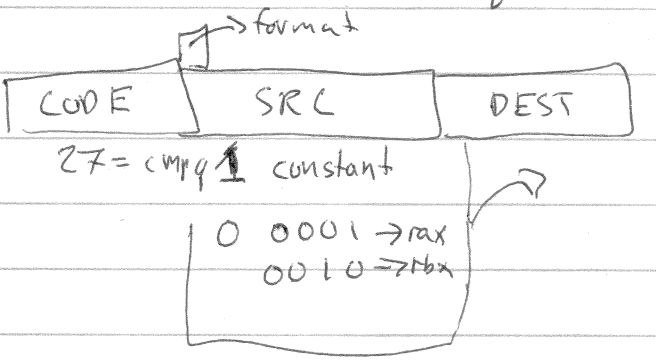


9-2

~~#t, #f => (set! name $0/$1)~~

(set! x #t) => (movq x $1)

(set! lhs
(cmp a1 a2)) => cmpq a2 a1
eq sete al
movzbq al, lhs




addq -8(%rbp), -16(%rbp)

---

$L_{before} = F(L_{after})$ $F(x) = (x - W(I)) \cup R(I)$

select-inst(if) = (if (si c) (si t) (si e))
(if c t e)

---

Liveness((if cmp a1 a2 (t...) (e...)), Lafter)
$L_t^B$ = Liveness((t...), Lafter)
$L_e^B$ = Liveness((e...), Lafter)

return $(L_t^B \cup L_e^B) \cup Vars(a_1) \cup Vars(a_2)$

$\{P\}$ (if cmp t e) $\{Q\}$
$\Leftarrow$ $\{P \wedge cmp\}$ t $\{Q\}$
$\{P \wedge \overline{cmp}\}$ e $\{Q\}$

q-3 | cmpq a2 a1)

↓ must be a reg or addr

cmpq $3, $4

↓

movq $3, %rax

cmpq %rax, $4

---

(if #t A B) => A  dead-code elimination DCE

(if (< 3 8) A B) => A  "partial-evaluation" / "constant-folding"

(if (not X) A B) => (if X B A)  GCC does, except when you say

(if (begin X Y) A B) => X; (if Y A B)

(if (begin X #t) A B) => X; A

X = new cons(1,2); => A

(if (new C $\vec{A}$) T F) => $\vec{A}$ ; T

(if C (let ([X a]) T) F) => (let ([X a]) (if C T F))

X ∉ FV(C) ∪ FV(F)

a has no effect

⇐