Lec 17 More Optimisations

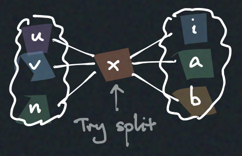
Splitting Live Ranges

Introduce moves to split temps
└ makes graph more sparse
Problem: no good and simple heuristic

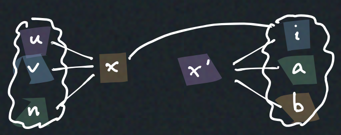
Consider:

```
    x ← y                                done:  a ← x' + 8128
    n ← u + v                                   b ← a + a
    i ← n                                       return x' * b
    x' ← x
l₁: if i ≤ 0 then l₂ else done
l₂: i ← i - 1
    x' ← x' * x'
    goto l₁
```

Observe x lives throughout func



Turns out splitting here is good

⇝ 

When to split: when dissatisfied with reg alloc result

Heuristics:
- split the one with longest live range?
- split in the middle?

Peephole Optimisation

← some 1000+ of these in LLVM

Only look a a few lines, make local optimisations

$$\left.\begin{array}{l} l: \ \dots \\ \vdots \\ l': \ \dots \end{array}\right\} \rightarrow \left\{\begin{array}{l} l: \ \dots \\ \vdots \\ l': \ \dots \end{array}\right.$$

▷ Constant folding

$l: x \leftarrow c_1 \otimes c_2 \ \} \rightarrow \{ \ l: x \leftarrow c$ if $c = c_1 \otimes c_2$

$l: x \leftarrow c_1 \otimes c_2 \ \} \rightarrow \{$ raise (arith) if $c_1 \otimes c_2$ undef

$l:$ if $c_1 \ ? \ c_2$ then l_1 else $l_2 \ \} \rightarrow \{ \ l:$ goto l_1 if $l_1 \ ? \ l_2$

One of them could be dead code

$$\left.\begin{array}{l} l_1: x \leftarrow y + c_1 \\ l_2: z \leftarrow x + c_2 \end{array}\right\} \rightarrow \left\{\begin{array}{l} l_1: x \leftarrow y + c_1 \\ l_2: z \leftarrow y + c \end{array}\right.$$ where $c = c_1 + c_2$

▷ Strength Reduction

$a + 0 = a$
$a - 0 = a$
$a * 0 = 0$
$a * 1 = a$
$a * 2^n = a << n$
$a * b + a * c = a * (b + c)$

⚠ $x + 1 \neq x$, etc.

▷ Null Sequences

$l: x \leftarrow x \ \} \rightarrow \{ \ l:$ nop

$$\left.\begin{array}{l} l_1: x \leftarrow y \\ l_2: y \leftarrow x \end{array}\right\} \rightarrow \left\{\begin{array}{l} l_1: x \leftarrow y \\ l_2: \text{nop} \end{array}\right.$$

▷ Useless goto

$$\left.\begin{array}{l} l_1: \text{goto } l_2 \\ l_2: \ \dots \end{array}\right\} \rightarrow \left\{\begin{array}{l} l_1: \text{nop} \\ l_2: \ \dots \end{array}\right.$$

Common Subexpression Elim

SSA

```
l : x ← s₁ ⊗ s₂                  l: x ← s₁ ⊗ s₂
  ⋮ ↶ x not written                 ⋮
      in btwn            ⇝
l': y ← s₁ ⊗ s₂                  l': y ← x
```

↶ if x defined & unchanged here
viz. every path reaching l' goes through l
"l dominates l'", "l > l'"

Ex.

```
lab₁:
   x ← a + b
   if a < 0 then lab₂ else lab₃

lab₂:                          lab₃:
   y ← a+b                        z ← a+b
   goto lab₄                      goto lab₄

lab₄:
```

~~n ← a+b~~ n ← x