Loop Invariant Code Motion

Loop invariant code motion

- Idea: some expressions evaluated in a loop never change; they are *loop invariant*
  - Can move loop invariant expressions outside the loop, store result in temporary and just use the temporary in each iteration
  - Why is this useful?
    - Think of this as CSE

Identifying loop invariant code

- To determine if a statement

  s: a = b op c

  is loop invariant, find all definitions of b and c that *reach* s

  - A statement t defining b reaches s if there is a path from t to s where b is not re-defined

- s is loop invariant if both b and c satisfy one of the following
  - it is constant
  - all definitions that reach it are from outside the loop
  - only one definition reaches it and that definition is also loop invariant

Moving loop invariant code

- Just because code is loop invariant doesn't mean we can move it!

```
for (...)
  a = b + c
```

```
do
 if (*)
  break
 a = 5
while (*)
c = a;
```

```
for (...)
 if (*)
  a = 5
 else
  a = 6
```

```
a = 5;
for (...)
 if (*)
  a = 4 + c
 b = a
```

- We can move a loop invariant statement a = b op c if
  - The statement dominates all loop exits where a is live
  - There is only one definition of a in the loop
  - a is not live before the loop
- Move instruction to a *preheader*, a new block put right before loop header

next: strength reduction