Lec 20 Function Optimisation

Loop optimisation through invarients (see slides)

1. Find loop invariants / induction variables
2. Hoist constants
3. Do other optimisations that are now possible
4. Eliminate accumulators

→ Loop unrolling — can enable one less mem access per iter

Tail Call Optimisation

Pow example

```
int powacc (int b, int e, int a) {
   if (e = 0) return a;
   else return powacc (b, e-1, a*b);
}

int pow (int b, int e) {
   return powacc (b, e, 1)
}
```

↑ potentially lots of stack frames

```
int powloop (int b, int e) {
   int acc = 0;
   while (e > 0) {
      e = e - 1;
      acc = acc * b;
   }
   return acc;
}
```

↑ one stack frame

For recursive function:

call f
ret ⇝ goto f

For nonrecursive tail calls ... more complicated

Function Inlining

Replace func call with func body

+ less stack frame, less moving, less reg saving
+ more optim opportunity
+ more flexibility in reg alloc

But we have to balance code size & perf gain

Troubles:
- recursive / mutually recursive funcs
- don't want to unroll a very frequently called func

Good situations to inline:
- func with small body
- func only called once