Code gen: functions

- Earlier discussed division of responsibilities between caller/callee

- Preserving/restoring registers
- Setting up/retrieving return values
- Setting up/cleaning up parameters
- Setting up/cleaning up local variables
- Transfer of control (where to jump to on call/return)

Optimizations

- compiler might inline functions that are only called once
- moving code to callee means it only appears in one spot in source code, instead of at each call/return point
- register preserve/restores may involve a lot of code, frequently repeated (if lots of registers used)
  - special hw instructions might save/restore many registers at once
  - caller might pass callee a list of which registers to save/restore, moving code into callee
  - compiler might generate special routine to save/restore (bypassing regular call/return rules since used by compiler only)

Parameter handling

- Pass-by-ref: ensure value is in memory at given address
- Pass-by-value: evaluate first, store in reg
- Source language may/may not specify order of evaluation of value parameters
- If unspecified, compiler should pick a consistent order, e.g. right-to-left, left-to-right
- also need to consider when side effects are applied in call

C example: what is the output?

```
f(int &a, int b, int c, int d, int e, int f, int g) {
    printf("%d %d %d %d %d %d %d\n", a,b,c,d,e,f,g);
    return a=a+a;
}

// then someplace in main
a=1;
a= f(a, a+=100, ++a, a-=20, a, a--, a);
printf("%d\n", a);
```

C example: continued

```
f(int &a, int b, int c, int d, int e, int f, int g) {
   printf("%d %d %d %d %d %d %d\n", a,b,c,d,e,f,g);
   return a=a++;
}
a=1;
a= f(a, a+=100, ++a, a-=20, a, a--, a);
printf("%d\n", a);
// f's output: 81 81 -19 20 0 1 1
// main's output: 81
// evaluating params right-to-left
// postincrement in return statement is "lost"
```

Implicit parameters

- compiler may add extra parameters, e.g.
  - pointer to callee's activation record
  - pointer to caller's activation record
  - return address
  - extra addressability values (e.g. Pointer to activation record for lexical parent in nested function declarations)
  - receiver parameter for OO (e.g. “this” pointer in C++)
  - function descriptors for functions as parameters
- depends on OS/hw stack/control mechanisms