Register Allocation Details

what to free?

- Some flexibility in this algorithm: when kicking a value out of a register, how do you decide which register to free?

- Lots of choices, different implications!
  - Ideal: want to kick out the value that causes the fewest downstream spills (e.g, will be used the least in the future)
  - Some simple tie-breaking ideas:
    - First, prefer to kick out non-dirty values
    - Second, prefer to kick out values that are going to be used farthest in the future

global variables

- Algorithms presented assume that loading a value into a register can be done in one instruction
- Loading value from global variables may take multiple instructions
    - Compute/load address for global: LA t1, <address of x>
    - Load from address into register: LW t2, 0(t1)

- Some options to handle:
    - Always allocate an extra register for global variable's address, only free it when value is freed (easy, but can waste registers)
    - Set aside register for address operations (even easier, but may require redundant address computations)
    - Treat loading address into register as another 3AC operation to process

Aliasing, as usual, is a problem

- What happens with this code?

```
//a and b are aliased
LD R1 a
LD R2 b
ADD R3 R1 R2 R3
ST R3 c // c = a + b
R1 = 7 //a = 7
ADD R4 R1 R2
ST R4 d // d = a + b
```

Dealing with aliasing

- Immediately before loading a variable x
    - For each variable aliased to x that is already in a dirty register, save it to memory (i.e., perform a store)
    - This ensures that we load the right value
- Immediately before writing to a register holding x
    - For each register associated with a variable aliased to x, mark it as invalid
    - So next time we use the variable, we will reload it
- Conservative approach: assume all variables are aliased (in other words, reload from memory on each read, store to memory on each write)
    - Better alias analysis can improve this

interaction

- Different optimizations interact with register allocation in different ways
    - Peephole optimizations can reduce register pressure, can make allocation better
    - CSE can actually increase register pressure (why?)
    - Different orders of optimization produce different results

- **Phase ordering** is an open problem in compilers

next: global register allocation