The Trouble with Aliasing

problem: aliasing

```
T1 = A * B
C = 7
T2 = A * B
```

What happens if C is aliased to B?

Aliasing

- One of the biggest problems in compiler analysis is to recognize aliases – different names for the same location in memory
- Aliases can occur for many reasons
  - Pointers referring to same location, arrays referencing the same element, function calls passing the same reference in two arguments, explicit storage overlapping (unions)
- Upshot: when talking about "live" and "killed" values in optimizations like CSE, we're talking about particular variable names
- In the presence of aliasing, we may not know which variables get killed when a location is written to

conservative approach

- Compiler optimization should be **sound**: should always generate correct code
  - A compiler should only perform an optimization if it *knows* it will not break the code
  - The opposite is not true! A compiler can choose *not* to perform an optimization, even if it is safe
- Sound approach in the case of pointers: assume worse-case scenario
  - All pointers point to the same location; all references are aliased
  - Writing to a variable kills *all* other variables that are references

Memory disambiguation

- Most compiler analyses rely on *memory disambiguation*
  - Otherwise, they need to be too conservative and are not useful
- Memory disambiguation is the problem of determining whether two references point to the same memory location
  - *Points-to* and *alias* analyses try to solve this
  - Will cover basic pointer analyses in a later lecture