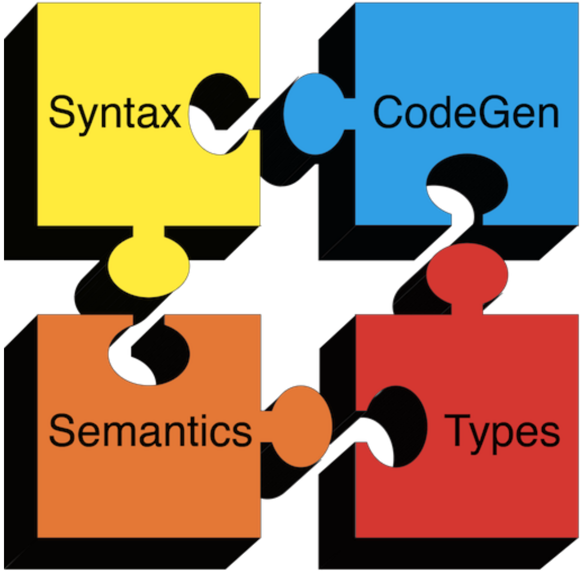


# Compilers

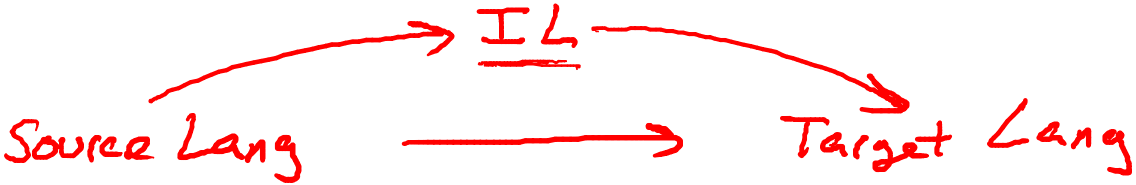
## Intermediate Code

Alex Aiken

# Intermediate Code

- A language between the source and the target

- Provides an intermediate level of abstraction
  - More details than the source
  - Fewer details than the target



# Intermediate Code

- Intermediate language = high-level assembly
  - Uses register names, but has an unlimited number
  - Uses control structures like assembly language
  - Uses opcodes but some are higher level
    - E.g., push translates to several assembly instructions
    - Most opcodes correspond directly to assembly opcodes

# Intermediate Code

- Each instruction is of the form

  x := y op z

  x := op y

  three-address code

  - y and z are registers or constants
  - Common form of intermediate code

- The expression x + (y * z) is translated

  $t_1 := y * z$

  $t_2 := x + t_1$

  - Each subexpression has a "name"

# Intermediate Code

- Similar to assembly code generation
- But use any number of IL registers to hold intermediate results

# Intermediate Code

- igen(e, t)
  - code to compute the value of e in register t

- Example:

  igen($e_1$ + $e_2$, t) =

  → igen($e_1$, $t_1$)  ($t_1$ is a fresh register)

  → igen($e_2$, $t_2$)  ($t_2$ is a fresh register)

  → $t := t_1 + t_2$

- Unlimited number of registers => simple code generation

# Intermediate Code

- You should be able to use intermediate code
  - At the level discussed in lectures

- You are not expected to know how to generate intermediate code
  - Because we won’t discuss it further
  - But really just a variation on code generation . . .