RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

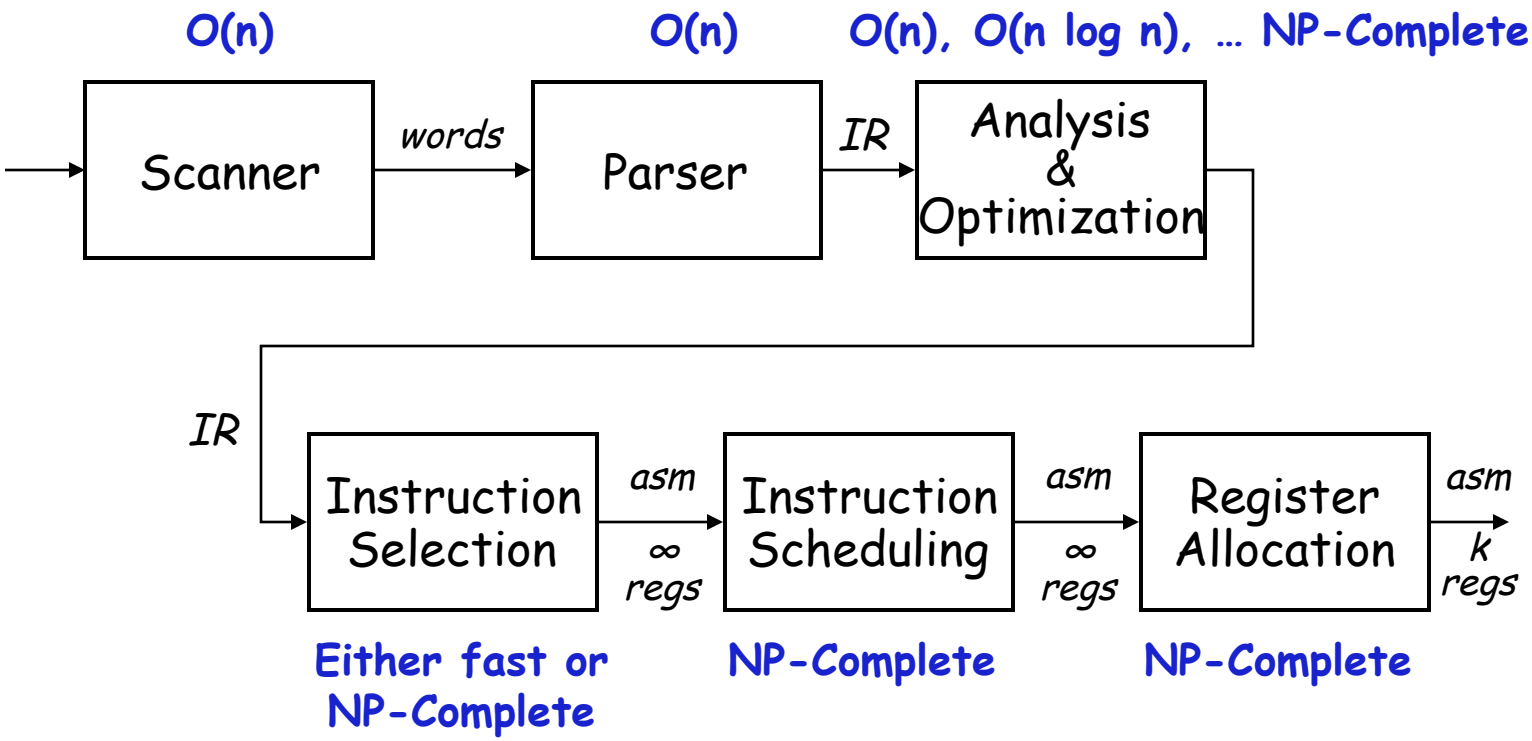
CS415 Compilers

Code Generation

These slides are based on slides copyrighted by Keith Cooper, Ken Kennedy & Linda Torczon at Rice University

Code Generation: Structure of a Compiler

(EaC Ch. 7)



A compiler is a lot of fast stuff followed by some hard problems

- → The hard stuff is mostly in code generation and optimization
- → For superscalars, its allocation & scheduling that is particularly important

Review - The Big Picture

Conventional wisdom says that we lose little by solving these problems independently

Note: many fuzzy terms here!

Instruction selection

- Use some form of pattern matching
- Assume enough registers or target "important" values

Instruction scheduling

- Within a block, list scheduling is "close" to optimal

Optimal for > 85% of blocks

- Across blocks, build framework to apply list scheduling

Register allocation

- Start from virtual registers & map into *k*
- Focus on good priority heuristic

Code Shape

Definition

- All those nebulous properties of the code that impact performance & code "quality"
- Includes code, approach for different constructs, cost, storage requirements & mapping, & choice of operations
- Code shape is the end product of many decisions *(big & small)*

Impact

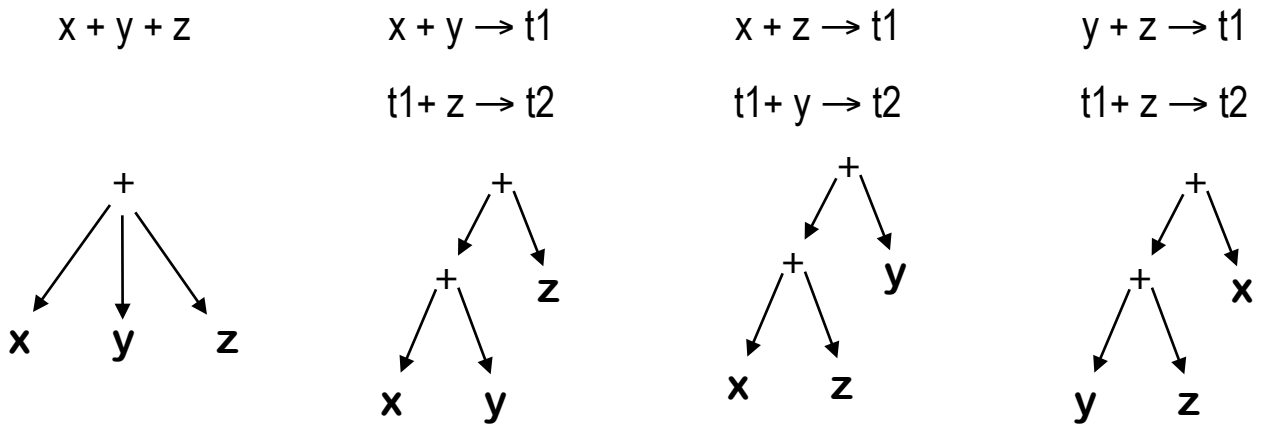
- Code shape influences algorithm choice & results
- Code shape can encode important facts, or hide them

Rule of thumb: expose as much derived information as possible

- Example: explicit branch targets in ILOC simplify analysis

Code Shape

An example



- What if x is 2 and z is 3?
- What if y+z is evaluated earlier?

Addition is commutative & associative for integers

The "best" shape for x+y+z depends on contextual knowledge

→ There may be several conflicting options

Code Shape

Another example -- the case statement

- Implement it as cascaded if-then-else statements
  - → Cost depends on where your case actually occurs
  - → O(number of cases)
- Implement it as a binary search
  - → Uniform (log n) cost
- Implement it as a jump table
  - → Lookup address in a table & jump to it
  - → Uniform (constant) cost

Compiler must choose best implementation strategy

No amount of massaging or transforming will convert one into another

Generating Code for Expressions

The key code quality issue is holding values in registers

- When can a value be safely allocated to a register?
  - → When only 1 name can reference its value (no aliasing)
  - → Pointers, parameters, aggregates & arrays all cause trouble
- When should a value be allocated to a register?
  - → When it is both *safe* & *profitable*

Encoding this knowledge into the *IR* (*register-register model*)

- Use code shape to make it known to every later phase
- Assign a virtual register to anything that can go into one
- Load or store the others at each reference

Relies on a strong register allocator

Next class

More code generation

Read EaC: Chapter 7.1 – 7.5