Lecture 6

EECS 483: COMPILER CONSTRUCTION

Announcements

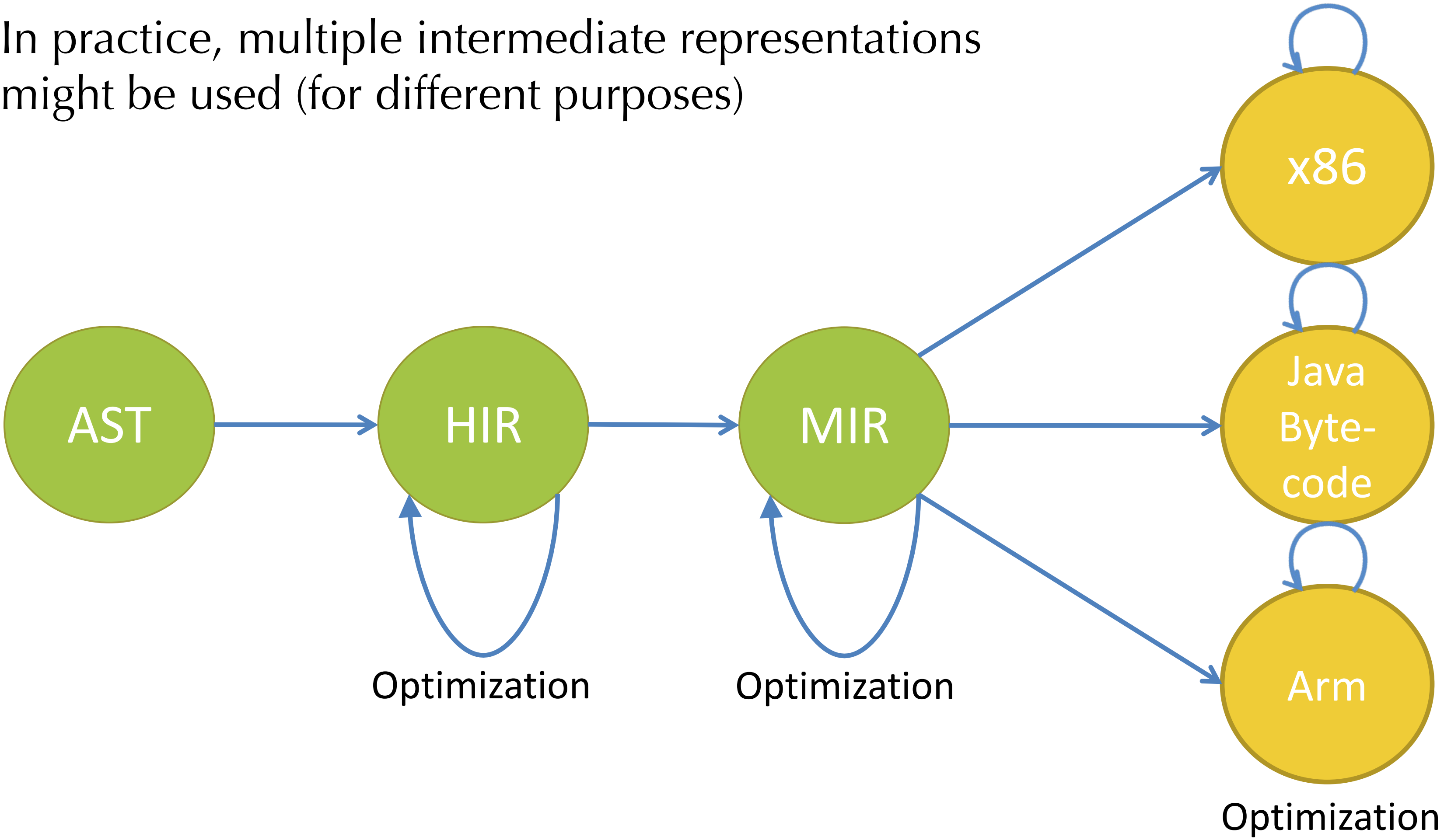
- HW2: X86lite
  - Available on the course web pages.
  - Due: Tuesday at 11:59:59pm
  - If you still haven't found a partner come up to the front after class.

see: ir-by-hand.ml, ir<X>.ml

INTERMEDIATE REPRESENTATIONS

Multiple IR's

- Goal: get program closer to machine code without losing the information needed to do analysis and optimizations
- In practice, multiple intermediate representations might be used (for different purposes)



Mid-level IR's: Many Varieties

- Intermediate between AST (abstract syntax) and assembly
- May have unstructured jumps, abstract registers or memory locations
- Convenient for translation to high-quality machine code
  - Example: all intermediate values might be named to facilitate optimizations that attempt to minimize stack/register usage

- Many examples:
  - **Triples**: OP a b
    - Useful for instruction selection on X86 via "tiling"
  - **Quadruples**: a = b OP c (RISC-like "three address form")
  - **Stack-based**:
    - Easy to generate
    - e.g., Java Bytecode, UCODE
  - **SSA**: variant of quadruples where each temporary is assigned exactly once
    - "pure" semantics (more like OCaml!)
    - Easy dataflow analysis for optimization
    - e.g., LLVM: industrial-strength IR, based on SSA

our destination

Intermediate Representations

- IR1: Expressions
  - *immutable* global variables
  - simple arithmetic *expressions*
- IR2: Commands
  - *mutable* global variables
  - *commands* for update and sequencing
- IR3: Local control flow
  - *conditional* commands & while *loops*
  - *basic blocks*
- IR4: Procedures (top-level functions)
  - *local variables*
  - *call stack*
- IR5: "almost" LLVM IR
  - missing *phi-nodes* (explained when we get there)

SSA is Functional Programming

- Explained in Andrew Appel's article: https://doi.org/10.1145/278283.278285
  - Variables are immutable
  - Jumps are analogous to tail calls
  - Makes analysis easier

- Most common way to compile imperative langauges is to use functional IRs!

- For today: all of our IRs today will be subsets of OCaml

Eliminating Nested Expressions

- Fundamental problem:
  – Compiling complex & nested expression forms to simple operations.

Source

```
((1 + X4) + (3 + (X1 * 5)))
```

AST

```
Add(Add(Const 1, Var X4),
    Add(Const 3, Mul(Var X1,
                     Const 5)))
```

IR

?

- Idea: *name* intermediate values, make order of evaluation explicit.
  – No nested operations.

Translation to SLL

- Given this:

```
Add(Add(Const 1, Var X4),
    Add(Const 3, Mul(Var X1,
                     Const 5)))
```

- Translate to this desired SLL form:

```
let tmp0 = add 1L varX4 in
let tmp1 = mul varX1 5L in
let tmp2 = add 3L tmp1 in
let tmp3 = add tmp0 tmp2 in
  tmp3
```

- Translation makes the order of evaluation explicit.
- Names intermediate values
- Note: introduced temporaries are never modified