Plan for Today

PA5: boolean expressions, control flow, and semantic analysis
- interpretation
- compilation to MIPS

Interpretation
- memory model in interpreter
- how to implement boolean expressions and control flow statements

Compilation to MIPS
- possible approaches including 3-address code as intermediate representation
- implicit stack-like intermediate representation

Generating MIPS code for control-flow and booleans

Example Program for PA5

```java
special main {
    int a;
    boolean b;
    a = 1;
    b = true && ! false && (a<3);

    if (b && true) {
        a = 42;
        System.out.println(1);
    } else {
        a = 7;
        System.out.println(0);
    }

    System.out.println(a*1 + 2);

    while (0 < a) {
        System.out.println(a);
        a = a - 10;
    }
}
```
Intermediate Program Representations

AST
– usually language dependent

Intermediate Representation (IR)
– Usually a language independent and target independent representation
– Examples
  – 3-address code
  – RTL used in GCC (like 3-address code)
  – LLVM used in the LLVM compiler (like 3-address code but typed)
  – Microsoft’s Common Intermediate Language (CIL)
  – Java byte code
  – Assem (an IR that wraps machine specific code)

AST ==> IR ==> target code

Intermediate Representations

Why?
– otherwise have to write MxN compilers instead of M front-ends and N backends
– want to do optimization on a generic representation

Desired characteristics of IRs
– should be easy to translate to
– should be easy to translate from to all target machines
– each piece should have simple semantics
– should be able to efficiently and effectively apply program optimizations
A Low-Level IR: 3-address code

3-address code
- Linear representation
- Typically language-independent
- Nearly corresponds to machine instructions

Example operations
- Copy \( x = z, t_1 = t_2 \)
- Unary op \( x = o p z \)
- Binary op \( x = v \ op \ z, t_1 = t_2 \ op \ t_3 \)
- Address of \( p = & v \)
- Load \( x = * p \)
- Store \( * p = x, \)
- Pass param \( \text{param} \ t_0 \)
- Call \( t_1 = \text{call} \ f, 1 \)
- Branch \( \text{goto} \ L_1 \)
- Cbranch \( \text{if (x==y)} \ \text{goto} \ L_1 \)

A stack-based IR

Stack-based IR
- Linear representation
- Language-independent
- Nearly corresponds to machine instructions

Memory model
- run-time stack, operations assume operands are on stack and result should be put on stack
- variables have memory locations

Example operations
- Push \( \text{push} \ \text{var}, \ \text{push} \ \text{constval} \)
- Pop \( \text{var} = \text{pop} \)
- Binary op \( \text{unop} \)
- Binary op \( \text{binop} \)
- Label \( L_1 \)
- Branch \( \text{goto} \ L_1 \)
- Cbranch \( \text{if} == \ \text{goto} \ L_1 \)