

Plan for Today

Bridging the semantic gap

- MiniJava to MIPS assembly

Intermediate Representations

- why?
- characteristics

3-address code

Tiger book expression Trees

Tiger book Assem representation

CS453 Lecture

Intermediate Representations

1

Bridging the semantic gap

```
class WhileUsage {
    public static void main(String[] a){
        System.out.println(new Foo().testing(5)); }
class Foo {
    public int testing(int p) {
        while (p<10) {
            System.out.println(p);
            p = p+1;
        }
        return 0; } }

```

```
main:      .text                                # push parameter onto stack
          subu $sp, $sp, 4
          # ExpCONST
          li t85, 5
          sw t85, 0($sp)
          # push parameter onto stack
          subu $sp, $sp, 4
          ...
          # push parameter onto stack
          subu $sp, $sp, 4
          # ExpCONST
          li t83, 4
          sw t83, 0($sp)
          jal _hallocc
          addu $sp, $sp, 4
          ...
          L6:
          # sink statement
          addu $sp, $sp, main_framsize
          lw $fp, -4($sp)
          j $ra
          ...

```

CS453 Lecture

Intermed

2

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
 - Tree data structure in the MiniJava Compiler (a little different)

AST ==> IR ==> target code

CS453 Lecture

Intermediate Representations

3

A Low-Level IR: 3-address code

3-address code

- Linear representation
- Typically language-independent
- Nearly corresponds to machine instructions

Example operations

- Assignment **x = y**
- Unary op **x = op y**
- Binary op **x = y op z**
- Address of **p = & y**
- Load **x = *p**
- Store **p = y**
- Pass param **param x_1**
- Call **y = call p, 1**
- Branch **goto L1**
- Cbranch **if (x==3) goto L1**

CS453 Lecture

Intermediate Representations

4

IR Code Generation

Goal

- Transforms AST into low-level *intermediate representation* (IR)

Simplifies the IR

- Removes high-level control structures: **for**, **while**, **do**, **switch**
- Removes high-level data structures: arrays, structs, unions, enums

Results in assembly-like code

- Semantic lowering
- Control-flow expressed in terms of “gotos”
- Each expression is very simple (three-address code)
e.g., $x = a * b * c$ \Rightarrow $t = a * b$
 $x = t * c$

CS453 Lecture

Intermediate Representations

5

Example

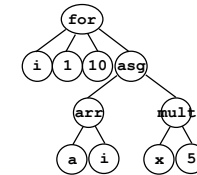
Source code (FORTRAN)

```
for i = 1 to 10 do  
  a(i) = x * 5;
```

Low-level IR (3-address code)

```
i := 1  
loop1:  
  if i > 10 goto loopexit  
  t1 = x * 5  
  t2 = &a  
  t3 = sizeof(int)  
  t4 = t3 * i  
  t5 = t2 + t4  
  *t5 = t1  
  i = i + 1  
  goto loop1  
loopexit:
```

High-level IR (AST)



CS453 Lecture

Intermediate Representations

6

Compiling Control Flow

Switch statements

- Convert **switch** into low-level IR

```
e.g., switch (c) {  
  case 0: f();  
         break;  
  case 1: g();  
         break;  
  case 2: h();  
         break;  
}  
  
if (c!=0) goto next1  
call f,0  
goto done  
next1: if (c!=1) goto next2  
call g,0  
goto done  
next2: if (c!=3) goto done  
call h,0  
done:
```

- Optimizations (depending on size and density of cases)
 - Create a jump table (store branch targets in table)
 - Use binary search

CS453 Lecture

Intermediate Representations

7

Compiling Arrays

Array declaration

- Store name, size, and type in symbol table

Array allocation

- Call `malloc()` or create space on the runtime stack

Array referencing (C is source code)

- e.g., $A[i]$ \Rightarrow $*(A + i * \text{sizeof}(A_elem))$

```
t2 = sizeof(A_elem)  
t3 = i * t2  
t4 = A + t3  
*t4
```

CS453 Lecture

Intermediate Representations

8

Missed Opportunities

3-address code is low level

- many architectures have CISC-like instructions
- the low-level IR might preclude using certain instructions in the ISA
- 6800 example

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

MiniJava Compiler Tree Language (Array Example)

`x[2] = 42;`

