Plan for Today

Bridging the semantic gap
  – MiniJava to MIPS assembly

Intermediate Representations
  – why?
  – characteristics

3-address code
Assem(MIPS) representation
Converting 3-address code to Assem(MIPS)

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

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:
.sub $fp, -4($sp)
move $fp, $sp
subu $sp, $sp, 8

# push parameter onto stack
subu $sp, $sp, 4
li t83, 4
sw t83, 0($sp)
jal _halloc
addu $sp, $sp, 4
...

L6:
# sink statement
addu $sp, $sp, main_framesize
lw $fp, -4($sp)
jal ra
...

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: \( x = y, t_1 = t_2 \)
- Indexed copy: \( x = y[i], x[i] = y, t_1 = y[i] \)
- Unary op: \( p = \# y \)
- Binary op: \( x = y \text{ op } z, t_1 = t_2 \text{ op } t_3 \)
- Load: \( x = \text{ *p} \)
- Store: \( \text{ *p} = y, \)
- Pass param: param \( x_1 \)
- Call: \( t_1 = \text{ call } f, 1 \)
- Branch: \( \text{ goto } L_1 \)
- Cbranch: \( \text{ if } (x==y) \text{ goto } L_1 \)

Assem intermediate representation

Assem.Instr
- “assembly language instruction without register assignments”

OPER(String assem, List<Temp> dst, List<Temp> src, List<Label> jumps)
- contains a string with holes for registers indicated by ‘d’ and ‘s’ and holes for labels indicated by ‘j’
- dst and src are lists of Temps whose register assignment should fill holes
- first entry in src is associated with ‘s0’, second with ‘s1’, etc.
- first entry in dst is associated with ‘d0’, etc.
- jumps is a list of labels for filling in label holes

Assem intermediate representation cont ...

LABEL(String assem, Label label)
- a label statement in the target code

MOVE(String assem, Temp dst, Temp src)
- similar to OPER in that assem string contains holes, but ..
  - no jumps
  - only one src and dst Temp

CJUMP(String a, Temp.Temp src1, RELOP op, Temp.Temp src2, Temp.Label t, Temp.Label f)
- similar to OPER in that assem string contains holes, but ..
  - only jumps to true and false target
  - only two source Temps for comparison
  - explicit conditional operation, which enables later changes in code layout

TempMap functionality in MipsFrame

static final Temp ZERO = new Temp(); // zero reg
static final Temp V0 = new Temp(); // function result
static final Temp T0 = new Temp(); // caller-saved
static final Temp T1 = new Temp();
... static final Temp SP = new Temp(); // stack pointer
static final Temp S8 = new Temp(); // callee-save (frame pointer)
static final Temp RA = new Temp(); // return address
private static final
HashMap<Temp,String> tempMap = new HashMap<Temp,String>(32);
static {
  tempMap.put(ZERO, "$0");
tempMap.put(V0, "$v0");
tempMap.put(T0, "$t0");
... tempMap.put(SP, "$sp");
tempMap.put(S8, "$s8");
tempMap.put(RA, "$ra");
}
public String tempMap(Temp temp) {
  if (tempMap.containsKey(temp)) {
    return tempMap.get(temp);
  } else {
    return temp.toString();
  }
}