

## Undergraduate Compilers Review cont...

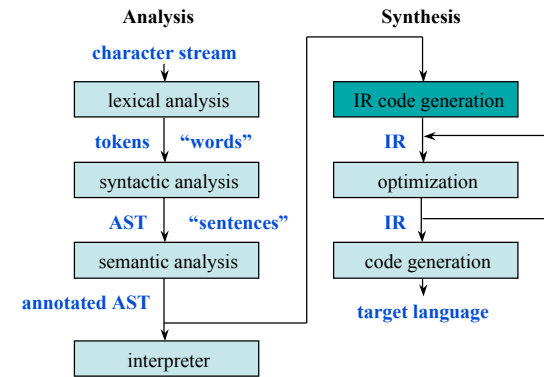
### Announcements

- Advice for the project writeups was posted on the mailing list
- SVN log will need more than one entry for a one day extension

### Today

- Generating intermediate representations
  - AST
  - 3-address code
  - Trees
  - Assem
- Generating MIPS assembly

## Structure of a Typical Compiler



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

AST ==> IR ==> target code

## 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.,  $x := a * b * c$   $\rightarrow$   $t := a * b$   
 $x := t * c$

## A Low-Level IR

### Register Transfer Language (RTL)

- 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+4)`
- Store `*(p+4) := y`
- Call `x := f()`
- Branch `goto L1`
- Cbranch `if (x==3) goto L1`

CS553 Lecture

Undergraduate Compilers Review

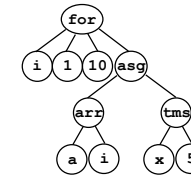
5

## Example

### Source code

```
for i = 1 to 10 do
  a[i] = x * 5;
```

### High-level IR (AST)



### Low-level IR (RTL)

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

CS553 Lecture

Undergraduate Compilers Review

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;
}
      next1: if (c!=1) goto next2
            g()
            goto done
      next2: if (c!=3) goto done
            h()
done:
```

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

CS553 Lecture

Undergraduate Compilers Review

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

- e.g., `A[i]` → `*(&A + i * sizeof(A_elem))`

```
t1 := &A
t2 := sizeof(A_elem)
t3 := i * t2
t4 := t1 + t3
*t4
```

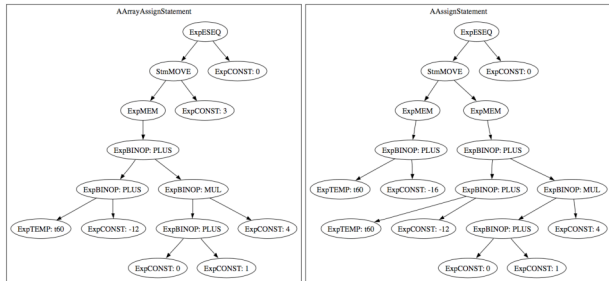
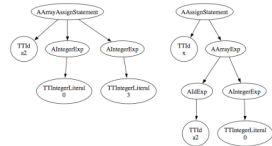
CS553 Lecture

Undergraduate Compilers Review

8

## MiniJava Compiler Tree Language (Array Example)

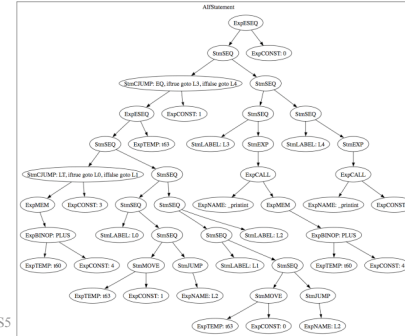
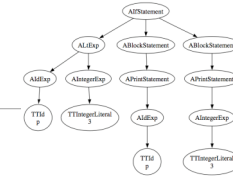
```
a2[0] = 3;
x = a2[0];
```



9

## MiniJava Compiler Tree Language (IF Example)

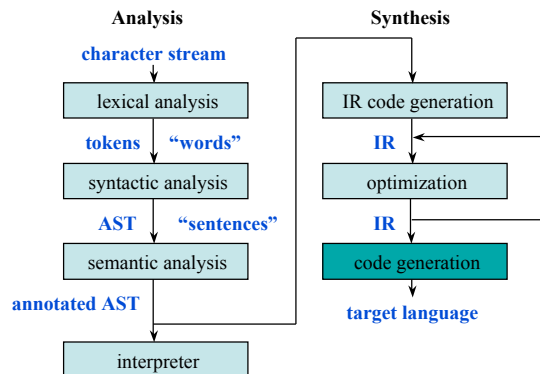
```
if (p<3) {
  System.out.println(p);
} else {
  System.out.println(3);
}
```



CS5

10

## Structure of a Typical Compiler



CS553 Lecture

Undergraduate Compilers Review

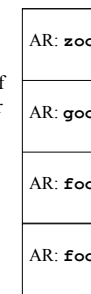
11

## Compiling Procedures

### Properties of procedures

- Procedures define scopes
- Procedure lifetimes are nested
- Can store information related to *dynamic invocation* of a procedure on a call stack (*activation record* or AR or stack frame):
  - Space for saving registers
  - Space for passing parameters and returning values
  - Space for local variables
  - Return address of calling instruction

higher addresses



lower addresses

stack

### Stack management

- Push an AR on procedure entry
- Pop an AR on procedure exit
- Why do we need a stack?

CS553 Lecture

Undergraduate Compilers Review

12

## Compiling Procedures (cont)

### Code generation for procedures

- Emit code to manage the stack
- Are we done?

### Translate procedure body

- References to local variables must be translated to refer to the current activation record
- References to non-local variables must be translated to refer to the appropriate activation record or global data space

CS553 Lecture

Undergraduate Compilers Review

13

## Code Generation

### Conceptually easy

- Three address code is a generic machine language
- Instruction selection converts the low-level IR to real machine instructions

### The source of heroic effort on modern architectures

- Alias analysis
- Instruction scheduling for ILP
- Register allocation
- More later. . .

CS553 Lecture

Undergraduate Compilers Review

14

## MIPS code generation in MiniJava compiler

### Assem data structure

- has string with source and destination spots to represent assembly instruction
- has list of uses, defs, and jump targets

### add rd, rs, rt

“add `d0, `s0, `s1”

### beq rs, rt, label

“beq `s0, `s0, `j0”

### lw rt, address

“lw `d0, #( `s0)”

### sw rt, address

“sw `s0, #( `s1)”

CS553 Lecture

Undergraduate Compilers Review

15

## Example MIPS program

```
class MultipleParams {
    public static void main(String[] a){
        System.out.println(new Foo().testing());}
}

class Foo {
    public int testing() {
        int local1;
        int local2;
        int local3;
        local1 = 1;
        local2 = 2;
        local3 = 3;
        return this.fooBar(local1, local2+local3,
            local3);
    }

    public int fooBar(int param1, int param2, int
        param3){
        return param1 - param2 * param3 ;
    }
}
```

```
.text
main:
main_frameSize=32
sw $fp, -4($sp)
move $fp, $sp

    subu $sp, $sp, main_frameSize
L1:
sw $ra, -8($fp)
...
L0:
# sink statement
addu $sp, $sp, main_frameSize
lw $fp, -4($sp)
j $ra

.text
testing:
testing_frameSize=64
sw $fp, -4($sp)
move $fp, $sp

    subu $sp, $sp, testing_frameSize
L3:

sw $ra, -8($fp)

li $t0, 1
sw $t0, -24($fp)

lw $t0, -24($fp)
sw $t0, -12($fp)
...
```

CS553 Lecture

Undergraduate Compilers Review

16

## Concepts

---

### Representations

- AST, low-level IR (RTL)

### Code Generation

- 3-address code
- IR Trees in MiniJava Compiler
  - Assumes infinite temporaries
- Mips
  - Requires mapping of all temporaries to an actual register

## Next Time

---

### Reading

- Ch 10

### Lecture

- Control Flow Graphs
- Liveness Analysis