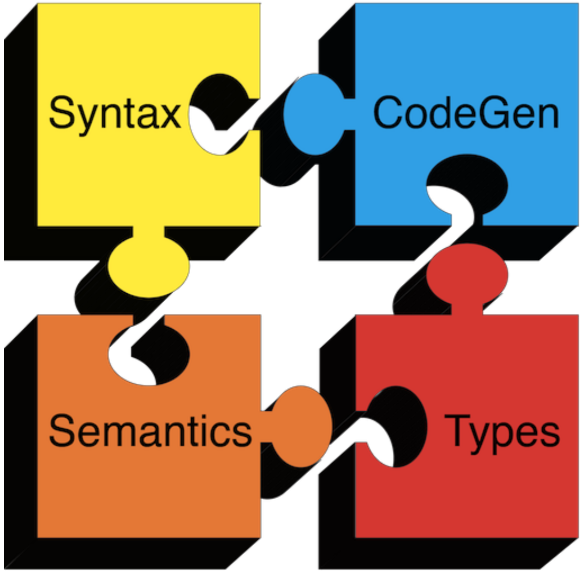

Compilers

Code Generation I

Alex Aiken

A language with integers and integer operations

$$P \rightarrow D; P \mid D$$

$$D \rightarrow \text{def id(ARGS)} = E;$$
$$\text{ARGS} \rightarrow \text{id, ARGS} \mid \text{id}$$

$$E \rightarrow \text{int} \mid \text{id} \mid \text{if } E_1 = E_2 \text{ then } E_3 \text{ else } E_4$$
$$\mid E_1 + E_2 \mid E_1 - E_2 \mid \text{id}(E_1,\ldots,E_n)$$

- The first function definition f is the entry point
  - The "main" routine

- Program for computing the Fibonacci numbers:

```
def fib(x) = if x = 1 then 0 else
             if x = 2 then 1 else
               fib(x - 1) + fib(x – 2)
```

- For each expression e we generate MIPS code that:
  - Computes the value of e in \$a0
  - Preserves \$sp and the contents of the stack

- We define a code generation function cgen(e) whose result is the code generated for e

- The code to evaluate a constant simply copies it into the accumulator:

  cgen(i) = li $a0 i

- This preserves the stack, as required

- Color key:
  - RED: compile time
  - BLUE: run time

$cgen(e_1 + e_2) =$

$cgen(e_1)$

```
sw $a0 0($sp)
addiu $sp $sp -4
```

$cgen(e_2)$

```
lw $t1 4($sp)
add $a0 $t1 $a0
addiu $sp $sp 4
```

$cgen(e_1 + e_2) =$

$cgen(e_1)$

print “sw $a0 0($sp)”

print “addiu $sp $sp -4”

$cgen(e_2)$

print “lw $t1 4($sp)”

print “add $a0 $t1 $a0”

print “addiu $sp $sp 4”

- Optimization: Put the result of e_1 directly in $t1?

cgen(e_1 + e_2) =

  cgen(e_1)

  move $t1 $a0

  cgen(e_2)

  add $a0 $t1 $a0

- The code for $+$ is a template with "holes" for code for evaluating e_1 and e_2

- Stack machine code generation is recursive
  - Code for $e_1 + e_2$ is code for e_1 and e_2 glued together

- Code generation can be written as a recursive-descent of the AST
  - At least for expressions

- New instruction: sub reg_1 reg_2 reg_3
  - Implements $reg_1 \leftarrow reg_2 - reg_3$

cgen(e_1 - e_2) =

```
cgen(e1)
sw $a0 0($sp)
addiu $sp $sp -4
cgen(e2)
lw $t1 4($sp)
sub $a0 $t1 $a0
addiu $sp $sp 4
```

Code Generation I

Choose the expression that the assembly code at right was generated from.

- ○ 5 + (4 - 3)
- ○ 5 - (4 + 3)
- ○ (5 + 4) - 3
- ○ (5 - 4) + 3

```
li $a0 5
sw $a0 0($sp)
addiu $sp $sp -4
li $a0 4
sw $a0 0($sp)
addiu $sp $sp -4
li $a0 3
lw $t1 4($sp)
sub $a0 $t1 $a0
addiu $sp $sp 4
lw $t1 4($sp)
add $a0 $t1 $a0
addiu $sp $sp 4
```

- New instruction: beq reg_1 reg_2 label
  - Branch to label if $reg_1 = reg_2$
- New instruction: b label
  - Unconditional jump to label

```
cgen(if e1 = e2 then e3 else e4) =
  cgen(e1)
  sw $a0 0($sp)
  addiu $sp $sp -4
  cgen(e2)
  lw $t1 4($sp)
  addiu $sp $sp 4
  beq $a0 $t1 true_branch
```

```
false_branch:
  cgen(e4)
  b end_if
true_branch:
  cgen(e3)
end_if:
```