



# COMP 181

---


## Lecture 15 Code generation II

November 7, 2005

## The class...



- The midterm was too long...
  - Yes, but it was the same length for everyone
  - We'll scale appropriately
- Lectures
  - I'll be out of town next week
  - Two options
    - A lecture by Noah
    - An extra lecture on a Friday – "fun" topic
- Homework
  - More frequent, smaller problem sets
- Projects
  - We'll still do two more
  - I'll scale them back



2

## Prelude


- Who are these people?
  - Pierre and Pam Omidyar
- And we care because....?
  - Pierre Omidyar is the founder of eBay
  - Donated \$100 million to Tufts
- But there are strings...
  - The money must be invested in *microfinance*
- What is microfinance?
  - Small business loans, typically given to individuals in developing countries
  - Average: \$600 (as low as \$40)

3

## Code generation

- Overview
  - Walk AST, generate code for each construct
  - Nested constructs:
    - Call generate on children as necessary
    - Use registers to pass values up tree
- Key: order is important
  - Emitted code goes into central list
  - Bottom-up post-order traversal

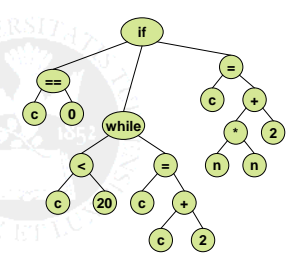



4

## Example

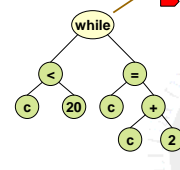
```

if (c == 0) {
  while (c < 20) {
    c = c + 2;
  }
}
else
  c = n * n + 2;
  
```

5


## Example



```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
  
```

L1: Code



6

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

**Code**

L1:

Tufts University Computer Science

7

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

```

t1 = generate(expr1)
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r

```

**Code**

L1:

Tufts University Computer Science

8

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

```

t1 = generate(expr1)
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r

```

```

r = new_temp() = R0
emit( r = load v )
return r

```

**Code**

L1:  
R0 = load c

Tufts University Computer Science

9

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

```

t1 = generate(expr1) = R0
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r

```

**Code**

L1:  
R0 = load c

Tufts University Computer Science

10

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

```

t1 = generate(expr1)=R0
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r

```

```

r = new_temp() = R1
emit( r = 20 )
return r

```

**Code**

L1:  
R0 = load c  
R1 = 20

Tufts University Computer Science

11

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

```

t1 = generate(expr1)=R0
t2 = generate(expr2)=R1
r = new_temp() = R2
emit( r = t1 op t2 )
return r

```

**Code**

L1:  
R0 = load c  
R1 = 20  
R2 = R0 < R1

Tufts University Computer Science

12

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

**Code**

```

L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0

```

13

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

**Code**

```

L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0

```

```

r = generate(expr2)
l = lgenerate(expr2)
emit( store *l = r )
return r

```

14

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

**Code**

```

L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2

```

```

r = generate(expr2)
l = lgenerate(expr2)
emit( store *l = r )
return r

```

```

t1 = generate(expr1)=R3
t2 = generate(expr2)=R4
r = new_temp() = R5
emit( r = t1 op t2 )
return r

```

15

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

**Code**

```

L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2

```

```

r = generate(expr2)=R5
l = lgenerate(expr2)
emit( store *l = r )
return r

```

16

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

**Code**

```

L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c

```

```

r = generate(expr2)=R5
l = lgenerate(expr2)
emit( store *l = r )
return r

```

```

r = new_temp() = R6
emit( r = & v )
return r

```

Something like:  
R6 = base + offset

17

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )

```

**Code**

```

L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5

```

```

r = generate(expr2)=R5
l = lgenerate(expr2)=R6
emit( store *l = r )
return r

```

18

### Example

```

E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
  
```

**Code**

```

L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5
goto L1
L0:
  
```

Tufts University Computer Science 19

### Example

```

Code
R7 = load c
R8 = 0
R9 = R7 == R8
not_goto R9,L3
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5
goto L1
L0:
goto L4
L3:
. . .
L4:
  
```

Tufts University Computer Science 20

### Nesting

```

Code
R7 = load c
R8 = 0
R9 = R7 == R8
not_goto R9,L3
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5
goto L1
L0:
goto L4
L3:
. . .
L4:
  
```

Tufts University Computer Science 21

### Code Shape

- Definition
  - All those nebulous properties of the code that impact performance & code "quality"
  - Includes:
    - Code for different constructs
    - Cost, storage requirements & mapping
    - Choice of operations
  - Code shape is the end product of many decisions
- Impact
  - Code shape influences algorithm choice & results
  - Code shape can encode important facts, or hide them

Tufts University Computer Science 22

### Code Shape

- An example:
  - $x + y + z$
  - $x + y \rightarrow t1$   
 $t1 + z \rightarrow t2$
  - $x + z \rightarrow t1$   
 $t1 + y \rightarrow t2$
  - $y + z \rightarrow t1$   
 $t1 + x \rightarrow t2$
- What if x is 2 and z is 3?
  - What if y+z is evaluated earlier?
- Addition is commutative & associative for integers
- The "best" shape for  $x+y+z$  depends on context  
There may be several conflicting options

Tufts University Computer Science 23

### Code Shape

- Another example – the switch statement
  - Implement it as a jump table
    - Lookup address in a table & jump to it
    - Uniform (constant) cost
  - Implement it as cascaded if-then-else statements
    - Cost depends on where your case actually occurs
    - $O(\text{number of cases})$
  - Implement it as a binary search
    - Uniform  $(\log n)$  cost
- Compiler must choose best implementation strategy  
No way to convert one into another

Tufts University Computer Science 24

## Order of evaluation

- Ordering for performance
  - Using associativity and commutativity
    - Very hard problem
  - Operands
    - op1 must be preserved while op2 is computed
    - Emit code for more intensive one first
- Language requirements
  - Sequence points:
    - Places where side effects must be visible to other operations
  - C examples:
 

<code>f() + g()</code>	may be executed in any order
<code>f()    g()</code>	f must be executed first
<code>f(i++)</code>	argument to f must be i+1



## Registers only

- Assume all variables are in registers
  - Add loads and stores later
  - Change *generate(v)*

```
r = new temp()
emit( r = v )
return r
```

```
Code
R7 = load c
R8 = 0
R9 = R7 == R8
not_goto R9, L3
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2, L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5
goto L1
L0:
goto L4
L3:
. . .
L4:
```



## Registers only

- Proliferation of registers and labels

Many CPUs have a fast `c == 0` test

Can use accumulators: `c = c + 2`

Label leads to another goto; may have multiple labels

```
Code
R7 = c
R8 = 0
R9 = R7 == R8
not_goto R9, L3
L1:
R0 = c
R1 = 20
R2 = R0 < R1
not_goto R2, L0
R3 = c
R4 = 2
R5 = R3 + R2
c = R5
goto L1
L0:
goto L4
L3:
. . .
L4:
```



## Efficient lowering

- Reduce number of temporary registers
  - Don't copy variable values into registers
  - Accumulate values, when possible
  - Reuse temporaries, where possible
- Generate more efficient labels
  - Don't generate multiple adjacent labels
  - Avoid goto-label-goto



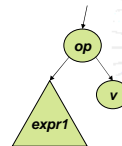
## Avoiding extra copies

- Basic algorithm
  - Recursive generation traverses to leaves
  - At leaves, generate: `R = v` or `R = c`
- Improvement
  - Stop recursion one level early
  - Check to see if children are leaves
  - Don't call generate recursively on variables, constants



## Avoiding copies

`expr1 op expr2`



```
if (expr1 is var)
  t1 = (Var) expr1
else
  t1 = generate(expr1)
if (expr2 is var)
  t2 = (Var) expr2
else
  t2 = generate(expr2)
r = new temp()
emit( r = t1 op t2 )
return r
```



## Example

- **Expr1** is (a+b)

- Not a leaf
- Recursively generate code
- Return temp

( a + b ) \* c

- **Expr2** is c

- Return c

### Code

```
R0 = a + b
R1 = R0 * c
```

- **Emit** ( R0 \* c )



## Use accumulation

- **Idea:**

- We only need 2 registers to evaluate  $\text{expr1 op expr2}$
- Reuse temp assigned to one of the subexpressions

```
if (expr1 is var)
  t1 = (Var) expr1
else
  t1 = generate(expr1)
if (expr2 is var)
  t2 = (Var) expr2
else
  t2 = generate(expr2)
emit( t1 = t1 op t2 )
return t1
```



## Example

- Combined:

- Remove copies
- Accumulate value
- Only need one register

( a + b ) \* c

- (Original version might have required as many as 6)

### Code

```
R0 = a + b
R0 = R0 * c
```



## Reuse of temporaries

- **Idea:**

$\text{expr1 op expr2}$  →

```
t1 = generate(expr1)
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r
```

- Can **generate**(expr1) and **generate**(expr2) share temporaries?

- Yes, except for t1 and t2
- Observation: temporaries have a limited lifetime
- Lifetime confined to a subtree

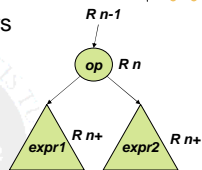


## Reuse of temporaries

- Subtrees can share registers

- Algorithm:

- Use a stack of registers
- Start at # = 0
- Each call to generate:
  - "Push" next number
  - Use any register > #
  - When done, "pop" back up



```
R# = expr1
R# = R# op expr2
```



## Next time...

- Introduction to optimization
- Later today: Project stage 3
- A small homework

