CSCI 742 - Compiler Construction

Lecture 29
Code Generation for Control Structures
Instructor: Hossein Hojjat

April 6, 2018
Recap: Code Generation for Expressions

\[
[e_1 + e_2] = \\
[e_1] \\
[e_2] \\
iadd
\]

\[
[e_1 \times e_2] = \\
[e_1] \\
[e_2] \\
imul
\]
Recap: Code Generation for Expressions

Code generation visits AST nodes in post-order

```plaintext
iconst_1
iconst_2
iadd
iconst_3
iconst_4
iadd
imul
```
Although JVM defines a boolean type, it only provides very limited support for it.

There are no JVM instructions solely dedicated to operations on boolean values.

Instead, expressions in Java that operate on boolean values are compiled to use values of int.

We represent Java boolean false in JVM by the integer 0.
We represent Java boolean true in JVM by the integer 1.
true, false, variables

- \([\text{true}] = \text{iconst}_1\)
- \([\text{false}] = \text{iconst}_0\)
- for boolean variable \(b\), for which \(n = \text{slotOf}(b)\)
- \([b] = \text{iload}_n\)
- \([b = e] = \)
  \[
  [e] 
  \text{istore}_n
  \]

Recap: if<cond> branches if int comparison with zero succeeds

\[
\begin{align*}
[\text{if (cond) } tStmt \text{ else } eStmt] &= \\
&= [\text{cond}] \\
&\quad \text{ifeq(nElse)} \\
&\quad [tStmt] \\
&\quad \text{goto(nAfter)} \\
\text{nElse: }&\quad [eStmt] \\
\text{nAfter: }&
\end{align*}
\]
Compiling `while` Statement

\[ \text{[while (cond) stmt]} = \]
\[
\text{nStart: } \ [\text{cond}]
\]
\[
\text{ifeq(nExit)}
\]
\[
\text{[stmt]}
\]
\[
\text{goto(nStart)}
\]
\[
\text{nExit: }
\]
Compiling while Statement

\[
\begin{align*}
\text{while } & (\text{cond}) \text{ stmt} = \\
n\text{Start} : & [\text{cond}] \\
& \text{ifeq}(n\text{Exit}) \\
& [\text{stmt}] \\
& \text{goto}(n\text{Start}) \\
n\text{Exit}: & \\
\end{align*}
\]

Exercise: Give a translation with only one jump during loop
Compiling while Statement

Exercise: Give a translation with only one jump during loop
Example: Code Generation for while Loop

```java
class Example {
    static boolean cond(int n) {
        /* ...*/
    }
    static int work(int n) {
        /* ...*/
    }
    static void func(int n) {
        while (cond(n)) {
            n = work(n);
        }
    }
}
```
Exercise

- Oberon-2 has a LOOP statement that expresses repetitions with exit condition in the middle of the loop
- This generalizes while and do ... while
- Give a translation scheme for the LOOP construct

```plaintext
LOOP
  code1
  EXIT IF cond
  code2
END
```
• Oberon-2 has a LOOP statement that expresses repetitions with exit condition in the middle of the loop
• This generalizes while and do ... while
• Give a translation scheme for the LOOP construct

```
LOOP
  code1
  EXIT IF  cond
  code2
END
```

```
nStart: [code1]
  [cond]
  ifneq(nExit)
  [code2]
  goto(nStart)

nExit:
```
Bitwise Operations

01001000 & 01001000  
10101110 = 10101110  
00001000 = 11101110

• **iand** computes the bitwise and of `value1` and `value2`  
  - (which must be ints)

• The int result replaces `value1` and `value2` on stack

![Diagram showing iand operation]

• **ior**: dual of **iand**
Compiling Bitwise Operations

\[
\begin{align*}
[e_1 \& e_2] &= [e_1] \quad [e_1 \mid e_2] &= [e_1] \\
&\quad [e_2] \quad &\quad [e_2] \\
i\text{and} & \quad i\text{or}
\end{align*}
\]
• Non-bitwise operators && and || are short-circuit operators in Java
• They only evaluate their second operand if necessary
• Must compile short-circuit operators correctly
• It is not acceptable to emit code that always evaluates both operands of && ,||

$$\left[ e_1 \&\& e_2 \right] = \left[ e_1 \right]$$
$$\left[ e_2 \right]$$

not allowed to evaluate $e_2$ if $e_1$ is false

Also for $(e_1||e_2)$: if $e_1$ true, $e_2$ not evaluated
• What does this program do?

```java
static boolean bigFraction(int x, int y) {
    return ((y == 0) | (x/y > 100));
}

public static void main(String[] args) {
    bigFraction(10, 0);
}
```

Exception in thread "main" java.lang.ArithmeticException: / by zero
• What does this program do?

```java
class Example {
    static boolean bigFraction(int x, int y) {
        return ((y == 0) | (x/y > 100));
    }
    public static void main(String[] args) {
        bigFraction(10, 0); // Should result in an exception
    }
}
```

• Exception in thread "main" java.lang.ArithmeticException: / by zero
Example

- What does this program do?

```java
static int iterate() {
    int[] x = new int[10];
    int i = 0;
    int res = 0;
    while ((i < x.length) & (x[i] >= 0)) {
        i = i + 1;
        res = res + 1;
    }
    return res;
}
```
Example

- What does this program do?

```java
static int iterate () {
    int [] x = new int [10];
    int i = 0;
    int res = 0;
    while ((i < x.length) && (x[i] >= 0)) {
        i = i + 1;
        res = res + 1;
    }
    return res ;
}
```

- Exception in thread "main"
  java.lang.ArrayIndexOutOfBoundsException: 10

should be `&&`
Conditional Expression

\[ c \ ? \ t \ : \ e \ \text{means:} \]

1. evaluate \( c \)
2. if \( c \) is true, then evaluate \( t \) and return
3. if \( c \) is false, then evaluate \( e \) and return

- To compile \(||, \&\&\) transform them into conditional expression

\[
(p \&\& q) \equiv (p) ? q : \text{false} \\
(p \| q) \equiv (p) ? \text{true} : q
\]
Compiling Conditional Expression

- Same as for if statement, even though code for branches will leave values on the stack

\[ [(\text{cond}) \; ? \; t \; : \; e] = \]

\[ \begin{align*}
&[(\text{cond})] \\
&\text{ifeq}(\text{nElse}) \\
&t \\
&\text{goto}(\text{nAfter}) \\
\text{nElse}: & [(e)] \\
\text{nAfter}: &
\end{align*} \]
Java Example for Conditional

```java
int f(boolean c, int x, int y) {
    return (c ? x : y);
}
```

```assembly
0: iload_1
1: ifeq 8
4: iload_2
5: goto 9
8: iload_3
9: ireturn
```
\[
((\text{cond}) \ ? \ t : e) = \\
\begin{cases} 
\text{cond} \\
\text{ifeq}(\text{nElse)} \\
[t] \\
goto(\text{nAfter}) 
\end{cases}
\]
\text{nElse:} \ [e] \\
\text{nAfter:} \\

\[
(p \ &\& \ q) = \\
\begin{cases} 
[p] \\
\text{ifeq}(\text{nElse)} \\
[q] \\
goto(\text{nAfter}) 
\end{cases}
\]
\text{nElse:} \ \text{iconst}_0 \\
\text{nAfter:}
\[
[(\text{cond}) \ ? \ t \ : \ e] = \begin{cases} 
[\text{cond}] 
\text{ifeq}(\text{nElse}) 
[t] 
\text{goto}(\text{nAfter}) 
\end{cases}
\]
\text{nElse:} \quad [e]
\text{nAfter:}

\[
[p \ || \ q] = \begin{cases} 
[p] 
\text{ifeq}(\text{nElse}) 
\text{iconst}_1 
\text{goto}(\text{nAfter}) 
\end{cases}
\]
\text{nElse:} \quad [q]
\text{nAfter:}