Recap: Code Generation for Expressions

\[
\left[ e_1 + e_2 \right] = \\
\left[ e_1 \right] \\
\left[ e_2 \right] \\
\text{iadd}
\]

\[
\left[ e_1 \cdot e_2 \right] = \\
\left[ e_1 \right] \\
\left[ e_2 \right] \\
\text{imul}
\]
Recap: Code Generation for Expressions

Code generation visits AST nodes in post-order

iconst_1
iconst_2
iadd
iconst_3
iconst_4
iadd
imul
JVM Boolean Type

- 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

Java Virtual Machine Specification
Java SE 8 Edition

- We represent Java boolean false in JVM by the integer 0
- We represent Java boolean true in JVM by the integer 1
• \([\text{true}] = \text{iconst}_1\)
• \([\text{false}] = \text{iconst}_0\)
• for boolean variable \(b\), for which \(n = \text{slotOf}(b)\)
• \([b] = \text{iload}_n\)
• \([b = e] =
  \[
  [e] \\
  \text{istore}_n
  \]
Recap: \( \text{if} < \text{cond} > \) branches if int comparison with zero succeeds

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

\[ [\text{while (cond) stmt}] = \]

\[ \text{nStart: } [\text{cond}] \]

\[ \text{ifeq(nExit)} \]

\[ [\text{stmt}] \]

\[ \text{goto(nStart)} \]

\[ \text{nExit: } \]
Compiling \texttt{while} Statement

\[\texttt{while} \ (\textit{cond}) \ \texttt{stmt} \ \texttt{K} \ \texttt{=} \ \texttt{nStart} : \ \texttt{cond} \ \texttt{K} \ \texttt{ifeq}(\texttt{nExit}) \ \texttt{K} \ \texttt{stmt} \ \texttt{K} \ \texttt{goto(} \texttt{nStart} \texttt{)} \ \texttt{K} \ \texttt{nExit} : \]

\textbf{Exercise:} Give a translation with only one jump during loop
Compiling \textbf{while Statement}

\[
\begin{aligned}
(\neg \text{cond}) &? \\
\text{stmt} &? \\
\text{nStart} & \\
\text{nExit} & \\
\text{nStmt} &
\end{aligned}
\]

\[
\begin{aligned}
[\text{while } (\text{cond}) \text{ stmt}] = \\
\text{nStart:} & \quad [\text{cond}] \\
\text{ifeq(nExit)} & \\
[\text{stmt}] & \\
\text{goto(nStart)} & \\
\text{nExit:} &
\end{aligned}
\]

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

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

```
0: iload_0
1: invokestatic #2 // cond:(I)Z
4: ifeq 15
7: iload_0
8: invokestatic #3 // work:(I)I
11: istore_0
12: goto 0
15: return
```
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

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

```
LOOP
  code1
  EXIT IF cond
  code2
END

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

nExit:
```
Bitwise Operations

\[
\begin{array}{c}
01001000 \& \\
10101110 =
\end{array}
\begin{array}{c}
00001000 \\
11101110
\end{array}
\]

- \textit{iand computes the bitwise and of value1 and value2} - (which must be ints)
- The int result replaces \textit{value1 and value2} on stack

- \textit{ior: dual of iand}
[e_1 \& e_2] =
[e_1]
[e_2]
\text{Iand}

[e_1 \lor e_2] =
[e_1]
[e_2]
\text{Ior}
Short-circuit Evaluation

- 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 `&&`, `||`.

\[
\begin{align*}
\llbracket e_1 \&\& e_2 \rrbracket &= \\
\llbracket e_1 \rrbracket &= \\
\llbracket e_2 \rrbracket &= \quad \text{not allowed to evaluate } e_2 \text{ if } e_1 \text{ is false}
\end{align*}
\]

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));
}

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

Exception in thread "main" java.lang.ArithmeticException: / by zero
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
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
Conditional Expression

c ? t : e 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

\[(p \&\& q) == (p) \, ? \, q \, : \, false\]
\[(p \, || \, q) == (p) \, ? \, true \, : \, q\]
• Same as for if statement, even though code for branches will leave values on the stack

\[
\begin{align*}
[(\text{cond}) \ ? \ t \ : \ e] &= \\
&= [\text{cond}] \\
&\quad \text{ifeq(nElse)} \\
&\quad [t] \\
&\quad \text{goto(nAfter)} \\
\text{nElse:} &\quad [e] \\
\text{nAfter:} &
\end{align*}
\]
int f(boolean c, int x, int y) {
    return (c ? x : y);
}
Compiling &&

\[
[(\text{cond}) \ ? \ t \ : \ e] = \\
[\text{cond}] \\
\text{ifeq(}\text{nElse}) \\
[t] \\
\text{goto(}\text{nAfter}) \\
\text{nElse: } [e] \\
\text{nAfter: }
\]

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

\[ [p \text{ || } q] = \]
\[
\begin{align*}
[(p) ? \text{true} : q] = \\
[p] \\
\text{ifeq}(\text{nElse}) \\
\text{iconst}_1 \\
\text{goto}(\text{nAfter}) \\
\text{nElse: } [q] \\
\text{nAfter:} \]
\]