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`

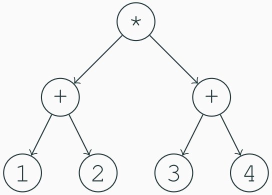
$[\![e_1 * e_2]\!] =$

$[\![e_1]\!]$

$[\![e_2]\!]$

`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

`true`, `false`, variables

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

  $[\![e]\!]$

  `istore_n`

Compiling `if` Statement

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

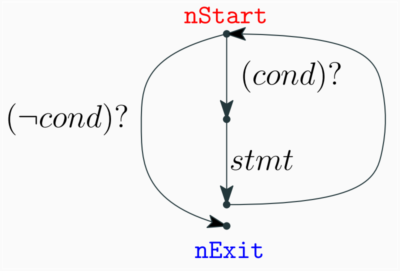
$[\![$`if` $(cond)$ $tStmt$ `else` $eStmt]\!] =$

```
        [[cond]]
        ifeq(nElse)
        [[tStmt]]
        goto(nAfter)
 nElse: [[eStmt]]
nAfter:
```

$[\![$`if` $(cond)$ $tStmt$ `else` $eStmt]\!] =$

```
        [[cond]]
        ifneq(nThen)
        [[eStmt]]
        goto(nAfter)
 nThen: [[tStmt]]
nAfter:
```

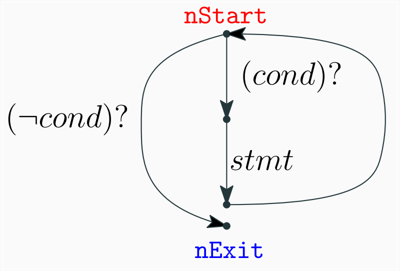
Compiling `while` Statement



$[\![$`while` $(cond)\ stmt]\!] =$

```
    nStart:  [[cond]]
             ifeq(nExit)
             [[stmt]]
             goto(nStart)
     nExit:
```

Compiling `while` Statement

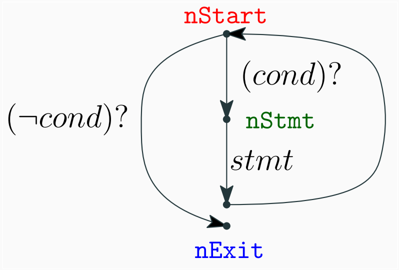


$[\![$`while` $(cond)$ $stmt]\!] =$

```
    nStart:  [[cond]]
             ifeq(nExit)
             [[stmt]]
             goto(nStart)
     nExit:
```

Exercise: Give a translation with only one jump during loop

Compiling `while` Statement



```
⟦while (cond) stmt⟧ =
   nStart:   ⟦cond⟧
             ifeq(nExit)
             ⟦stmt⟧
             goto(nStart)
    nExit:
```

```
⟦while (cond) stmt⟧ =
                   goto(nStart)
           nStmt:  ⟦stmt⟧
          nStart:  ⟦cond⟧
                   ifneq(nStmt)
```

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

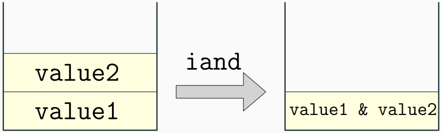
```
LOOP
  code1
  EXIT IF cond
  code2
END
```

```
nStart:  ⟦code1⟧
         ⟦cond⟧
         ifneq(nExit)
         ⟦code2⟧
         goto(nStart)
 nExit:
```

Bitwise Operations

```
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



- `ior`: dual of `iand`

Compiling Bitwise Operations

$$
\begin{aligned}
[\![e_1 \,\&\, e_2]\!] = {} & \\
& [\![e_1]\!] \\
& [\![e_2]\!] \\
& \texttt{iand}
\end{aligned}
\qquad\qquad
\begin{aligned}
[\![e_1 \mid e_2]\!] = {} & \\
& [\![e_1]\!] \\
& [\![e_2]\!] \\
& \texttt{ior}
\end{aligned}
$$

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 $\&\&$,||

$$\xcancel{\begin{aligned} &[\![e_1 \mathbin{\&\&} e_2]\!] = \\ &\qquad [\![e_1]\!] \\ &\qquad [\![e_2]\!] \\ &\qquad \ldots \end{aligned}}$$

$\longleftarrow$ 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

Example

- What does this program do?

```
static boolean bigFraction(int x, int y) {
  return ((y==0) | (x/y > 100));
}
public static void main(String[] args) {
  bigFraction(10,0);
}
```

Example

- What does this program do?

```
static boolean bigFraction(int x, int y) {
  return ((y==0) | (x/y > 100));
}
public static void main(String[] args) {
  bigFraction(10,0);
}
```

should be ||

- 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?

```
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;
}
```

should be &&

- 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

$$(p \ \&\& \ q) == (p) \ ? \ q \ : \texttt{false}$$
$$(p \ || \ q) == (p) \ ? \ \texttt{true} \ : q$$

Compiling Conditional Expression

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

$$[\![(cond)\ ?\ t\ :e]\!] =$$

```
         [[cond]]
         ifeq(nElse)
         [[t]]
         goto(nAfter)
 nElse:  [[e]]
nAfter:
```

Java Example for Conditional

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

```
0: iload_1
1: ifeq 8
4: iload_2
5: goto 9
8: iload_3
9: ireturn
```

Compiling &&

$[\![(cond)\ ?\ t\ :\ e]\!] =$

```
        ⟦cond⟧
        ifeq(nElse)
        ⟦t⟧
        goto(nAfter)
nElse:  ⟦e⟧
nAfter:
```

$[\![p\ \&\&\ q]\!] =$

$[\![(p)\ ?\ q\ :\ \texttt{false}]\!] =$

```
        ⟦p⟧
        ifeq(nElse)
        ⟦q⟧
        goto(nAfter)
nElse:  iconst_0
nAfter:
```

Compiling ||

$$\llbracket (cond)\ ?\ t\ :\ e \rrbracket =$$

```
         ⟦cond⟧
         ifeq(nElse)
         ⟦t⟧
         goto(nAfter)
nElse:   ⟦e⟧
nAfter:
```

$$\llbracket p \mid\mid q \rrbracket =$$

$$\llbracket (p)\ ?\ \texttt{true}\ :\ q \rrbracket =$$

```
         ⟦p⟧
         ifeq(nElse)
         iconst_1
         goto(nAfter)
nElse:   ⟦q⟧
nAfter:
```