Code Generation

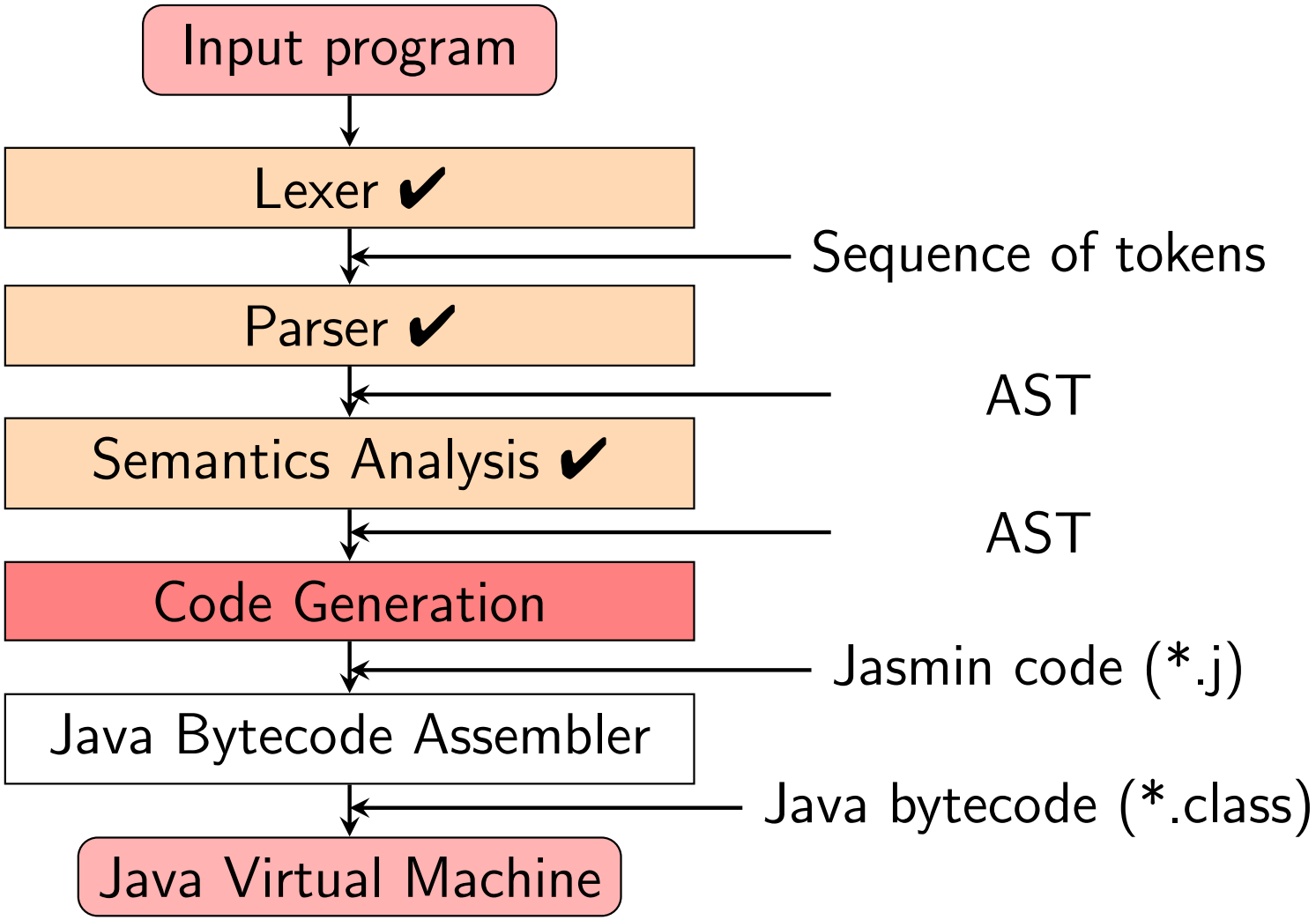
Dr. Phung Nguyen

Faculty of Computer Science and Engineering
University of Technology
HCMC Vietnam National University

10, 2015

Outline

Our Compiler



Remind

What is the appropriate Java bytecode of the following expression?
a = b + 1 //Assume that a is allocated at index 1 and b at 2

A. iload_1
iload_2
iconst_1
iadd

B. iload_2
iconst_1
iadd
istore_1

C. iload_1
iconst_1
iadd
istore_2

D. istore_1
iload_2
iadd
iconst_1

BKOOL-Java mapping

- A source program $\Rightarrow$ Java class(es)
- A class $\Rightarrow$ a class
- An instance method $\Rightarrow$ an instance method
- A static method $\Rightarrow$ a static method
- A parameter $\Rightarrow$ a parameter
- A local variable $\Rightarrow$ a local variable
- An expression $\Rightarrow$ an expression
- A statement $\Rightarrow$ a statement
- An invocation $\Rightarrow$ an invocation

Some issues

- An array declaration $\Rightarrow$ a declaration + code
  - ◁ class field $\Rightarrow$code in a class init
  - ◁ instance field $\Rightarrow$code in an init
  - ◁ local $\Rightarrow$code in the enclosing method
- A main method $\Rightarrow$a main method with different signature
  - ◁ void static main() $\Rightarrow$public static void main(String [] args)

Example

```
class Main {
   a:integer;
   final integer b = 1;
   c:float[3];
   static d:float;

   void main() {
       d:float[3];
       x: Main;
       x := new Main();
       x.c[0] := d[0];
       while (x.a > 10) do {
           x.a := x.a + 1;
       }
   }
}
```

```
public class Main {
    protected int a;
    protected final int b = 1;
    protected float c[] = new float[3];
    public static float d;

    public static void main(String[] arg) {
        float d[] = new float[3];
        Main x = new Main()
        x.c[0] = d[0];
        while (x.a > 10) do {
            x.a = x.a + 1;
        }
      return ;
    }
}
```

Jasmin

- Java source code to Java byte code

```
javac -g <*.java>
```

  -g: to generate debug information

- Java byte code to Jasmin code

```
java -cp <pathTo bcel>/bcel.jar JasminVisitor <*.class>
```

  Read JavaToJasmin script for details

- Jasmin code to Java byte code

```
java  -jar <pathTo jasmin>/jasmin.jar <*.j>
```

  Read run script for details

Example-Jasmin

```
public class Main {
      protected int a;
      protected final int b = 1;
      protected float c[] =
                  new float[3];
      public static float d;
```

Example-Jasmin

```
public class Main {
      protected int a;
      protected final int b = 1;
      protected float c[] =
                  new float[3];
      public static float d;
```

```
.source Main.java
.class public Main
.super java/lang/Object
.field protected a I
.field protected final b I = 1
.field protected c [F
.field static public d F
```

Example-Jasmin

```
public class Main {
      protected int a;
      protected final int b = 1;
      protected float c[] =
                  new float[3];
      public static float d;
```

```
.source Main.java
.class public Main
.super java/lang/Object
.field protected a I
.field protected final b I = 1
.field protected c [F
.field static public d F
```

```
.method public <init>()V
.limit stack 2
.limit locals 1
.var 0 is this LMain; from Label0 to Label1
Label0: aload_0
        invokespecial java/lang/Object/<init>()V
        aload_0
        iconst_1
        putfield Main.b I
        aload_0
        iconst_3
        newarray float
        putfield Main.c [F
Label1: return
.end method
```

Example-Jasmin (cont'd)

```
public static void main
                (String[] arg) {
    float d[] = new float[3];
    Main x = new Main();
```

```
.method public static main
                ([Ljava/lang/String;)V
.limit stack 4
.limit locals 3
.var 0 is arg [Ljava/lang/String;
        from Label1 to Label2
.var 1 is d [F
        from Label3 to Label2
.var 2 is x LMain;
        from Label5 to Label2
```

Example-Jasmin (cont'd)

```
public static void main
                (String[] arg) {
    float d[] = new float[3];
    Main x = new Main();
```

```
.method public static main
                ([Ljava/lang/String;)V
.limit stack 4
.limit locals 3
.var 0 is arg [Ljava/lang/String;
        from Label1 to Label2
.var 1 is d [F
        from Label3 to Label2
.var 2 is x LMain;
        from Label5 to Label2
```

```
float d[] = new float[3];
```

```
iconst_3
newarray float
astore_1
```

Example-Jasmin (cont'd)

```
public static void main
                (String[] arg) {
    float d[] = new float[3];
    Main x = new Main();
```

```
.method public static main
                ([Ljava/lang/String;)V
.limit stack 4
.limit locals 3
.var 0 is arg [Ljava/lang/String;
        from Label1 to Label2
.var 1 is d [F
        from Label3 to Label2
.var 2 is x LMain;
        from Label5 to Label2
```

```
float d[] = new float[3];
```

```
iconst_3
newarray float
astore_1
```

```
Main x = new Main();
```

```
new Main
dup
invokespecial Main/<init>()V
astore_2
```

Example-Jasmin (cont'd)

```
x.c[0] = d[0];
```

```
aload_2
getfield Main.c [F
iconst_0
aload_1
iconst_0
faload
fastore
```

Example-Jasmin (cont'd)

```
x.c[0] = d[0];
```

```
aload_2
getfield Main.c [F
iconst_0
aload_1
iconst_0
faload
fastore
```

```
while (x.a > 10) do {
    x.a = x.a + 1;
} ;
```

```
aload_2
getfield Main.a I
bipush 10
if_icmple Label0
aload_2
aload_2
getfield Main.a I
iconst_1
iadd
putfield Main.a I
Label0:
```

Example-Jasmin (cont'd)

```
x.c[0] = d[0];
```

```
aload_2
getfield Main.c [F
iconst_0
aload_1
iconst_0
faload
fastore
```

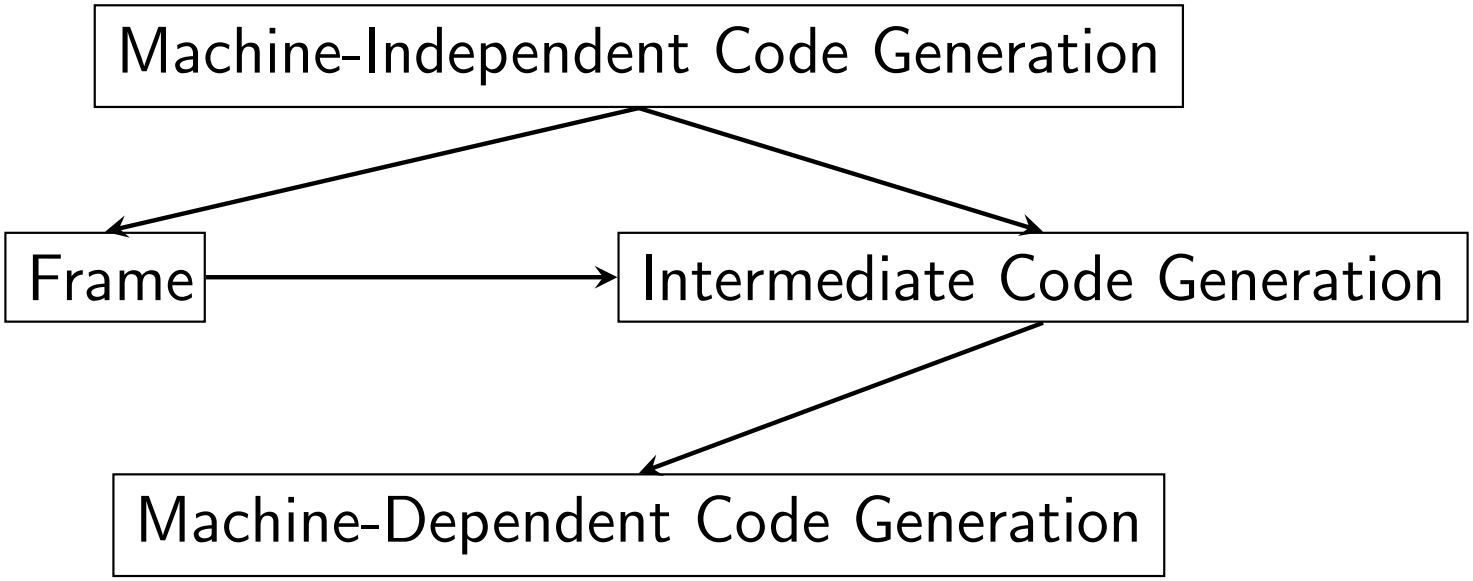
```
while (x.a > 10) do {
    x.a = x.a + 1;
} ;
```

```
aload_2
getfield Main.a I
bipush 10
if_icmple Label0
aload_2
aload_2
getfield Main.a I
iconst_1
iadd
putfield Main.a I
Label0:
```

```
return;
```

```
return
.end method
```

Code Generation Design



Machine-Dependent Code Generation

- Generating specified machine code
  E.g.: emitLDC(20) $\rightarrow$ ldc 20
- Implemented in JasminCode

Intermediate Code Generation

- Depend on both language and machine

Intermediate Code Generation

- Depend on both language and machine
- Select instructions

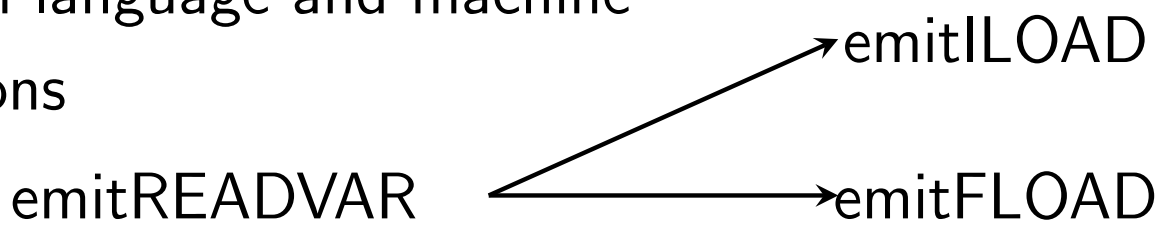
Intermediate Code Generation

- Depend on both language and machine
- Select instructions

emitREADVAR → emitILOAD

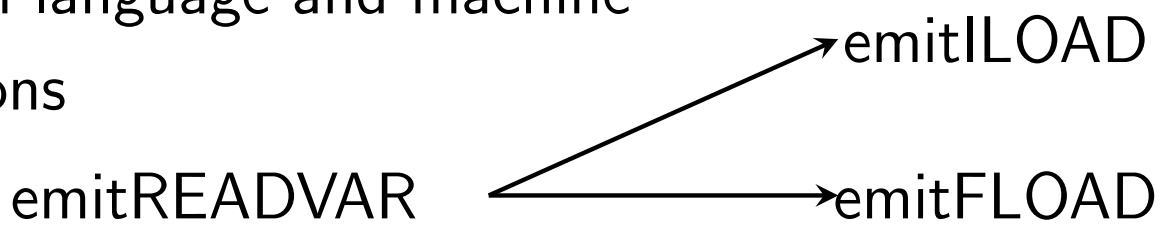
Intermediate Code Generation

- Depend on both language and machine
- Select instructions



Intermediate Code Generation

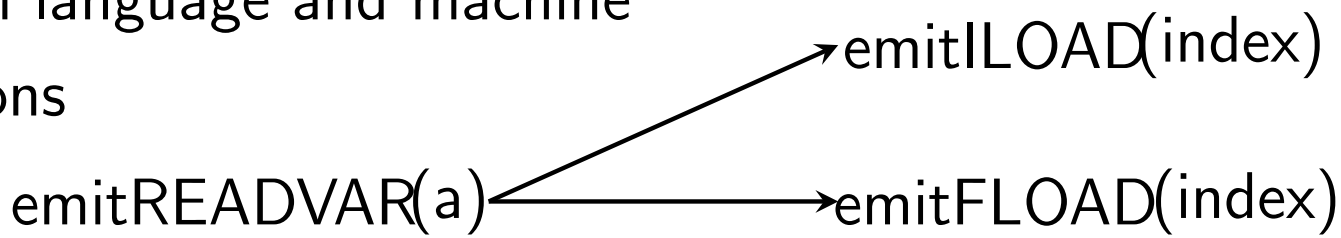
- Depend on both language and machine
- Select instructions



- Select data objects

Intermediate Code Generation

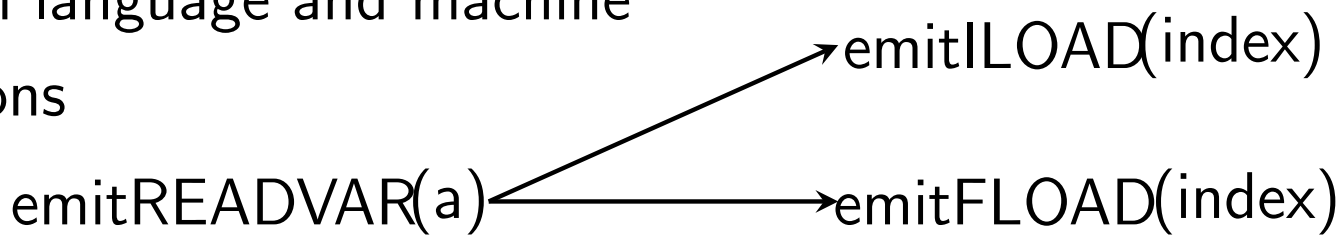
- Depend on both language and machine
- Select instructions



- Select data objects

Intermediate Code Generation

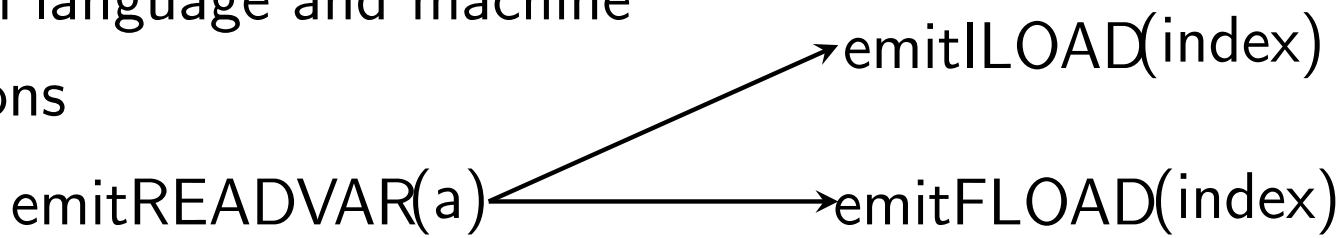
- Depend on both language and machine
- Select instructions



- Select data objects

Intermediate Code Generation

- Depend on both language and machine
- Select instructions



- Select data objects
- Simulate the execution of the machine

Intermediate Code Generation

- Depend on both language and machine
- Select instructions

emitREADVAR(a) → emitILOAD(index)

emitREADVAR(a) → emitFLOAD(index)

- Select data objects
- Simulate the execution of the machine
  - ◁ emitICONST → push()

Intermediate Code Generation

- Depend on both language and machine
- Select instructions

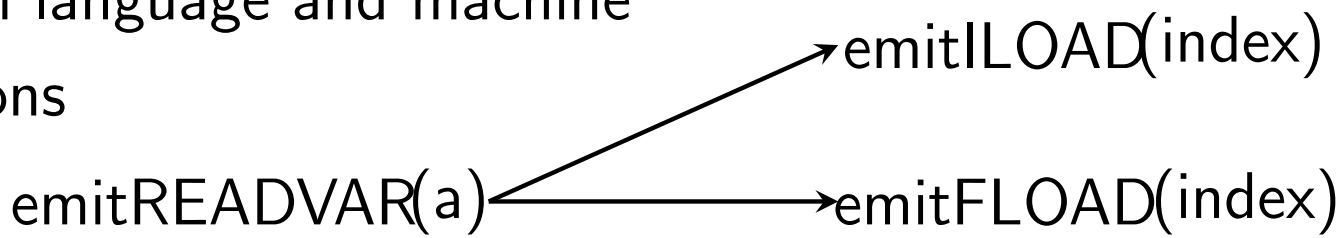
emitREADVAR(a) → emitILOAD(index)

emitREADVAR(a) → emitFLOAD(index)

- Select data objects
- Simulate the execution of the machine
  - ◁ emitICONST → push()
  - ◁ emitISTORE → pop()

Intermediate Code Generation

- Depend on both language and machine
- Select instructions



- Select data objects
- Simulate the execution of the machine
  - ◁ emitICONST → push()
  - ◁ emitISTORE → pop()
- Implemented in class Emitter

Frame

Tools are used to manage information used to generate code for a method

- isMain: generating code for main is different than doing for a normal method
- Labels: are valid in the body of a method
  - getNewLabel(): return a new label
  - getStartLabel(): return the beginning label of a scope
  - getEndLabel(): return the end label of a scope
  - getContinueLabel(): return the label where a continue should come
  - getBreakLabel(): return the label where a break should come
  - enterScope()
  - exitScope()
  - enterLoop()
  - exitLoop()

Frame (cont'd)

- Local variable array
  - getNewIndex(): return a new index for a variable
  - getMaxIndex(): return the size of the local variable array
- Operand stack
  - push(): simulating a push execution
  - pop(): simulating a pop execution
  - getMaxOpStackSize(): return the max size of the operand stack
- Implemented in class Frame

Machine-Independent Code Generation

- Based on the source language
- Use facilities of Frame and Intermediate Code Generation (Emitter)

Outline

Symbol Entries for Local

- String name;
- Type type;
- Kind kind; //Variable, Constant
- Option[String] obj;
  - ◁ the index if id is a local variable
  - ◁ the value of constant

Declarations

- Class Fields
  .field static name type-desc
  - ◁ for an array, generating code to initialize the array in the class init
- Instance Fields
  .field name type-desc
  - ◁ for an array, generating code to initialize the array in the init
  - ◁ for a constant, generating code to initialize the constant in the init
- Local variables
  .var var-index is name type-desc scopeStart-label scopeEnd-label
  - ◁ for an array, generating code to initialize the array in the method
  - ◁ for a constant, generating code to initialize the constant in the method

Method Declarations

- Method Implementation
  .method public class-name/method-name type-desc
  .end method
- How ?

Method Declarations

- Method Implementation
  .method public class-name/method-name type-desc
  .end method
- How ?
  Emitter.emitMETHOD(class-name/method-name, type)
  Emitter.emitENDMETHOD()

Outline

Expressions

- Literals
- Arithmethic expressions
- Boolean expressions
- Relational expressions
- Variables
- Field Access
- Array Cell
- Call

Integer Literals

- Machine code
  - ◁ iconst_m1, iconst_0, iconst_1, ...
  - ◁ iconst <num >
  - ◁ bipush <num >
  - ◁ sipush <num >
  - ◁ ldc <num >
- How ?

Integer Literals

- Machine code
  - ◁ iconst_m1, iconst_0, iconst_1, ...
  - ◁ iconst <num >
  - ◁ bipush <num >
  - ◁ sipush <num >
  - ◁ ldc <num >
- How ?
  emitter.emitICONST(literal,frame)

Arithmetic Expression

```
x.a + 1;
```

```
aload_2
getfield Main.a I
iconst_1
iadd
```

Arithmetic Expression

```
x.a + 1;
```

```
aload_2
getfield Main.a I
iconst_1
iadd
```

- generate code for operands and then operator
- BinaryOp(op,left,right) $\Rightarrow$

Arithmetic Expression

```
x.a + 1;
```

```
aload_2
getfield Main.a I
iconst_1
iadd
```

- generate code for operands and then operator
- BinaryOp(op,left,right) $\Rightarrow$
  - ◁ visit(ast.left)// generate code for left operand
  - ◁ visit(ast.right) // generate code for right operand
  - ◁ generate code for operators

Relational Expression

$$a > b$$

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

Relational Expression

- code for left expr.

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

Relational Expression

- code for left expr.
- code for right expr.

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

- code for left expr.
- code for right expr.
- get 2 new labels

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

- code for left expr.
- code for right expr.
- get 2 new labels
- emitRELOP(op,label 1)

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

- code for left expr.
- code for right expr.
- get 2 new labels
- emitRELOP(op,label 1)
- code "iconst_1"

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

- code for left expr.
- code for right expr.
- get 2 new labels
- emitRELOP(op,label 1)
- code "iconst_1"
- code goto + label 2

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

- code for left expr.
- code for right expr.
- get 2 new labels
- emitRELOP(op,label 1)
- code "iconst_1"
- code goto + label 2
- code emitLabel(label 1)

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

- code for left expr.
- code for right expr.
- get 2 new labels
- emitRELOP(op,label 1)
- code "iconst_1"
- code goto + label 2
- code emitLabel(label 1)
- code "iconst_0"

Relational Expression

$a > b$

```
    iload_2
    iload_3
    if_icmple Label0
    iconst_1
    goto Label1
Label0:
    iconst_0
Label1:
```

- code for left expr.
- code for right expr.
- get 2 new labels
- emitRELOP(op,label 1)
- code "iconst_1"
- code goto + label 2
- code emitLabel(label 1)
- code "iconst_0"
- code emitLabel(label 2)

Boolean Expressions

- Simple: like arithmetic expression

Boolean Expressions

- Simple: like arithmetic expression
- Short-circuit evaluation:

Boolean Expressions

- Simple: like arithmetic expression
- Short-circuit evaluation:
  - ◁ generate code for the left expression

Boolean Expressions

- Simple: like arithmetic expression
- Short-circuit evaluation:
  - ◁ generate code for the left expression
  - ◁ generate code to check the result of the left expression

Boolean Expressions

- Simple: like arithmetic expression
- Short-circuit evaluation:
  - ◁ generate code for the left expression
  - ◁ generate code to check the result of the left expression
  - ◁ generate code for the right expression

Boolean Expressions

- Simple: like arithmetic expression
- Short-circuit evaluation:
  - ◁ generate code for the left expression
  - ◁ generate code to check the result of the left expression
  - ◁ generate code for the right expression

Boolean Expressions

- Simple: like arithmetic expression
- Short-circuit evaluation:
  - ◁ generate code for the left expression
  - ◁ generate code to check the result of the left expression
  - ◁ generate code for the right expression

```
(b > 1 && b < 10)
```

```
        iload_2
        iconst_1
        if_icmple Label0
        iload_2
        bipush 10
        if_icmplt Label1
Label0:
```

Outline

Statements

- Block Statements
- Assignment Statements
- If Statements
- While Statements
- Do Statements
- For Statements
- Continue and Break Statements
- Return Statements
- Call Statements

Block Statements

```
{ int a;
  a = 1;
}
{ boolean b;
  b = true;
}
```

```
.var 2 is a I from Label3 to Label4
Label3:
    iconst_1
    istore_2
Label4:
.var 2 is b Z from Label5 to Label6
Label5:
    iconst_1
    istore_2
Label6:
```

Block Statements

```
{ int a;
  a = 1;
}
{ boolean b;
  b = true;
}
```

```
.var 2 is a I from Label3 to Label4
Label3:
    iconst_1
    istore_2
Label4:
.var 2 is b Z from Label5 to Label6
Label5:
    iconst_1
    istore_2
Label6:
```

- frame.enterScope

Block Statements

```
{ int a;
  a = 1;
}
{ boolean b;
  b = true;
}
```

```
.var 2 is a I from Label3 to Label4
Label3:
    iconst_1
    istore_2
Label4:
.var 2 is b Z from Label5 to Label6
Label5:
    iconst_1
    istore_2
Label6:
```

- frame.enterScope
- generate code for declaration list in the block

Block Statements

```
{ int a;
  a = 1;
}
{ boolean b;
  b = true;
}
```

```
.var 2 is a I from Label3 to Label4
Label3:
    iconst_1
    istore_2
Label4:
.var 2 is b Z from Label5 to Label6
Label5:
    iconst_1
    istore_2
Label6:
```

- frame.enterScope
- generate code for declaration list in the block
- generate the label of the beginning of the block

Block Statements

```
{ int a;
  a = 1;
}
{ boolean b;
  b = true;
}
```

```
.var 2 is a I from Label3 to Label4
Label3:
    iconst_1
    istore_2
Label4:
.var 2 is b Z from Label5 to Label6
Label5:
    iconst_1
    istore_2
Label6:
```

- frame.enterScope
- generate code for declaration list in the block
- generate the label of the beginning of the block
- generate the code of the block

Block Statements

```
{ int a;
  a = 1;
}
{ boolean b;
  b = true;
}
```

```
.var 2 is a I from Label3 to Label4
Label3:
    iconst_1
    istore_2
Label4:
.var 2 is b Z from Label5 to Label6
Label5:
    iconst_1
    istore_2
Label6:
```

- frame.enterScope
- generate code for declaration list in the block
- generate the label of the beginning of the block
- generate the code of the block
- generate the label of the end of the block

Block Statements

```
{ int a;
  a = 1;
}
{ boolean b;
  b = true;
}
```

```
.var 2 is a I from Label3 to Label4
Label3:
    iconst_1
    istore_2
Label4:
.var 2 is b Z from Label5 to Label6
Label5:
    iconst_1
    istore_2
Label6:
```

- frame.enterScope
- generate code for declaration list in the block
- generate the label of the beginning of the block
- generate the code of the block
- generate the label of the end of the block
- frame.exitScope

Assignment Statements

Assign(LHS,Expr)

Assignment Statements

Assign(LHS,Expr)

- Id
- Field Access
- Array Cell

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
        iload_1
        iload_2
        if_icmple Label1
        iconst_1
        goto Label2
    Label1:
        iconst_0
    Label2:
        ifeq Label3
        iconst_1
        istore_3
        goto Label4
    Label3:
        iconst_2
        istore_3
    Label4:
```

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
        iload_1
        iload_2
        if_icmple Label1
        iconst_1
        goto Label2
    Label1:
        iconst_0
    Label2:
        ifeq Label3
        iconst_1
        istore_3
        goto Label4
    Label3:
        iconst_2
        istore_3
    Label4:
```

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

- code for expr

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

- code for expr
- get 2 new labels

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

- code for expr
- get 2 new labels
- code "ifeq" + label1

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

- code for expr
- get 2 new labels
- code "ifeq" + label1
- code for "then" stmt

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

- code for expr
- get 2 new labels
- code "ifeq" + label1
- code for "then" stmt
- code "goto" + label2

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

- code for expr
- get 2 new labels
- code "ifeq" + label1
- code for "then" stmt
- code "goto" + label2
- code "label1"

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

- code for expr
- get 2 new labels
- code "ifeq" + label1
- code for "then" stmt
- code "goto" + label2
- code "label1"
- code for "else" stmt

If Statements

```
if (a > b)
    c = 1;
else
    c = 2;
```

```
    iload_1
    iload_2
    if_icmple Label1
    iconst_1
    goto Label2
Label1:
    iconst_0
Label2:
    ifeq Label3
    iconst_1
    istore_3
    goto Label4
Label3:
    iconst_2
    istore_3
Label4:
```

- code for expr
- get 2 new labels
- code "ifeq" + label1
- code for "then" stmt
- code "goto" + label2
- code "label1"
- code for "else" stmt
- code "label2"

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

- get 2 new labels

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

- get 2 new labels
- code "label1"

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

- get 2 new labels
- code "label1"
- code for expr

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

- get 2 new labels
- code "label1"
- code for expr
- code "ifeq" + label2

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

- get 2 new labels
- code "label1"
- code for expr
- code "ifeq" + label2
- code for "do" stmt

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

- get 2 new labels
- code "label1"
- code for expr
- code "ifeq" + label2
- code for "do" stmt
- code "goto" + label1

While Statements

```
while (a > b) do
        a = a - 1;
```

```
Label1:
    iload_1
    iload_2
    if_icmple Label3
    iconst_1
    goto Label4
Label3:
    iconst_0
Label4:
    ifeq Label2
    iload_1
    iconst_1
    isub
    istore_1
    goto Label1
Label2:
```

- get 2 new labels
- code "label1"
- code for expr
- code "ifeq" + label2
- code for "do" stmt
- code "goto" + label1
- code "label2"

Break and Continue Statements

When generating code for a body of a loop, how to generate code for a **break** or **continue** statement?

Break and Continue Statements

When generating code for a body of a loop, how to generate code for a **break** or **continue** statement?

- Frame.enterLoop
  - create 2 new labels **continueLabel** and
  - breakLabel
  - put these labels onto **continueStack** and **breakStack**
  - **Frame.getContinueLabel** and **Frame.getBreakLabel** return these labels

Break and Continue Statements

When generating code for a body of a loop, how to generate code for a **break** or **continue** statement?

- Frame.enterLoop
  - create 2 new labels **continueLabel** and
  - breakLabel
  - put these labels onto **continueStack** and **breakStack**
  - **Frame.getContinueLabel** and **Frame.getBreakLabel** return these labels
- Frame.exitLoop
  - pop these labels out of **continueStack** and **breakStack**

Break and Continue Statements

When generating code for a body of a loop, how to generate code for a **break** or **continue** statement?

- Frame.enterLoop
  - create 2 new labels **continueLabel** and
  - breakLabel
  - put these labels onto **continueStack** and **breakStack**
  - **Frame.getContinueLabel** and **Frame.getBreakLabel** return these labels
- Frame.exitLoop
  - pop these labels out of **continueStack** and **breakStack**

Break and Continue Statements

When generating code for a body of a loop, how to generate code for a **break** or **continue** statement?

- Frame.enterLoop
  - ◁ create 2 new labels **continueLabel** and
  - ◁ breakLabel
  - ◁ put these labels onto **continueStack** and **breakStack**
  - ◁ **Frame.getContinueLabel** and **Frame.getBreakLabel** return these labels
- Frame.exitLoop
  - ◁ pop these labels out of **continueStack** and **breakStack**
- continue; ⇒ emitter.emitGOTO(frame.getContinueLabel)
- break; ⇒ emitter.emitGOTO(frame.getBreakLabel)

Revised While Statements

```
while (a > b) do
    a = a - 1;
```

```
Label1:
iload_1
iload_2
if_icmple Label3
iconst_1
goto Label4
Label3:
iconst_0
Label4:
ifeq Label2
iload_1
iconst_1
isub
istore_1
goto Label1
Label2:
```

Revised While Statements

```
while (a > b) do
    a = a - 1;
```

```
Label1:
iload_1
iload_2
if_icmple Label3
iconst_1
goto Label4
Label3:
iconst_0
Label4:
ifeq Label2
iload_1
iconst_1
isub
istore_1
goto Label1
Label2:
```

- frame.enterLoop

Revised While Statements

```
while (a > b) do
    a = a - 1;
```

```
Label1:
iload_1
iload_2
if_icmple Label3
iconst_1
goto Label4
Label3:
iconst_0
Label4:
ifeq Label2
iload_1
iconst_1
isub
istore_1
goto Label1
Label2:
```

- frame.enterLoop
- get breakLabel and continueLabel

Revised While Statements

```
while (a > b) do
    a = a - 1;
```

```
Label1:
iload_1
iload_2
if_icmple Label3
iconst_1
goto Label4
Label3:
iconst_0
Label4:
ifeq Label2
iload_1
iconst_1
isub
istore_1
goto Label1
Label2:
```

- frame.enterLoop
- get breakLabel and continueLabel
- code continueLabel

Revised While Statements

```
while (a > b) do
    a = a - 1;
```

```
Label1:
iload_1
iload_2
if_icmple Label3
iconst_1
goto Label4
Label3:
iconst_0
Label4:
ifeq Label2
iload_1
iconst_1
isub
istore_1
goto Label1
Label2:
```

- frame.enterLoop
- get breakLabel and continueLabel
- code continueLabel
- code for expr

Revised While Statements

```
while (a > b) do
    a = a - 1;
```

```
Label1:
iload_1
iload_2
if_icmple Label3
iconst_1
goto Label4
Label3:
iconst_0
Label4:
ifeq Label2
iload_1
iconst_1
isub
istore_1
goto Label1
Label2:
```

- frame.enterLoop
- get breakLabel and continueLabel
- code continueLabel
- code for expr
- code "ifeq" + breakLabel

Revised While Statements

```
while (a > b) do
    a = a - 1;
```

```
Label1:
iload_1
iload_2
if_icmple Label3
iconst_1
goto Label4
Label3:
iconst_0
Label4:
ifeq Label2
iload_1
iconst_1
isub
istore_1
goto Label1
Label2:
```

- frame.enterLoop
- get breakLabel and continueLabel
- code continueLabel
- code for expr
- code "ifeq" + breakLabel
- code for "do" stmt

Revised While Statements

```
while (a > b) do
    a = a - 1;
```

```
Label1:
iload_1
iload_2
if_icmple Label3
iconst_1
goto Label4
Label3:
iconst_0
Label4:
ifeq Label2
iload_1
iconst_1
isub
istore_1
goto Label1
Label2:
```

- frame.enterLoop
- get breakLabel and continueLabel
- code continueLabel
- code for expr
- code "ifeq" + breakLabel
- code for "do" stmt
- code "goto" + continueLabel

Revised While Statements

```
while (a > b) do
    a = a - 1;
```

```
Label1:
iload_1
iload_2
if_icmple Label3
iconst_1
goto Label4
Label3:
iconst_0
Label4:
ifeq Label2
iload_1
iconst_1
isub
istore_1
goto Label1
Label2:
```

- frame.enterLoop
- get breakLabel and continueLabel
- code continueLabel
- code for expr
- code "ifeq" + breakLabel
- code for "do" stmt
- code "goto" + continueLabel
- code breakLabel

Do and For Statements

Do and For Statements

Do it yourself

Call Statements

Call Statements

Look in the initial code

Outline

Jasmin Directives

Jasmin Directives

Look in the initial code

Summary

- Use BCEL to know which code should be generated

Summary

- Use BCEL to know which code should be generated
- Generate code for expressions first

Summary

- Use BCEL to know which code should be generated
- Generate code for expressions first
- Generate code for statements later

Summary

- Use BCEL to know which code should be generated
- Generate code for expressions first
- Generate code for statements later
- Good luck