SWEN430 - Compiler Engineering

Lecture 12 - Bytecode Generation

Erin Greenwood-Thessman

with thanks to Linsday Groves & David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Java Bytecode Example

```
class Test {
 public int f(int x) {
  int y = x * 2;
  return y + x;
}}
```

```
public int f(int);
  Code:
   Stack=2, Locals=3
   0:    iload_1
   1:    iconst_2
   2:    imul
   3:    istore_2
   4:    iload_2
   5:    iload_1
   6:    iadd
   7:    ireturn
```

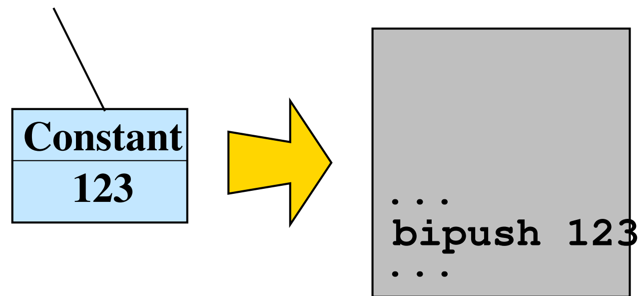
How do we get from AST to bytecode?

https://en.wikipedia.org/wiki/List_of_Java_bytecode_instructions lists bytecodes in a concise table.

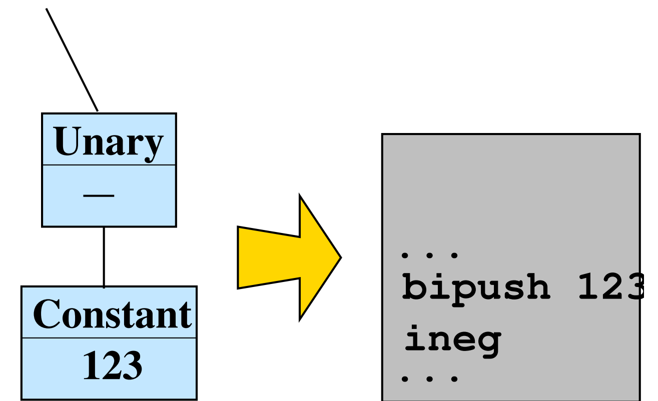
https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html JVM Spec.

Basic Calculations

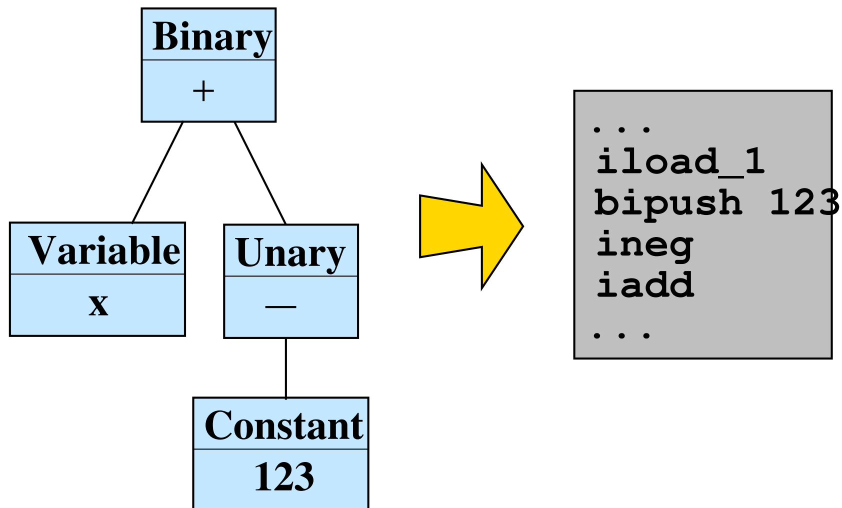
1.



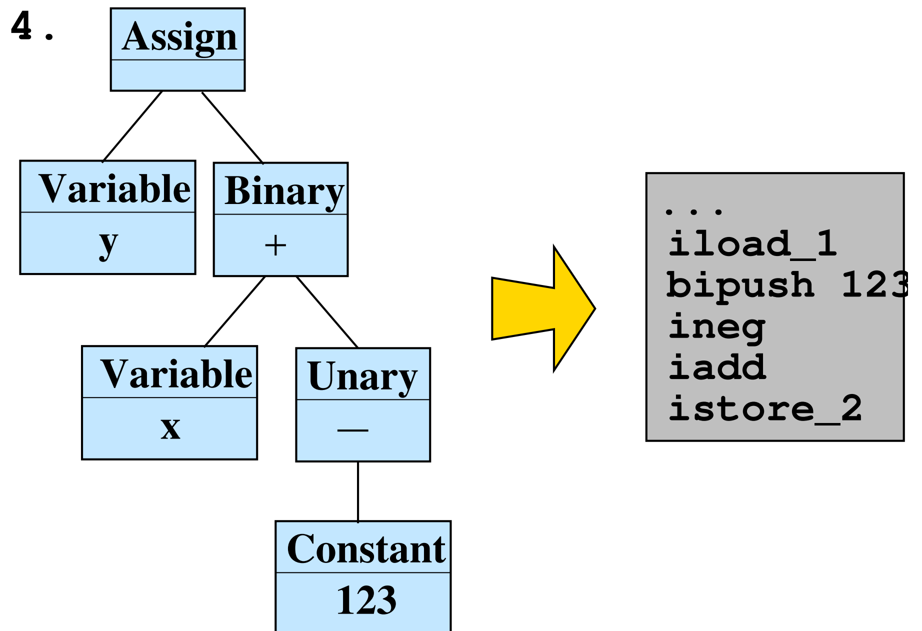
2.



3.



4.



Generating Simple Bytecodes

- Often several choices of bytecode:

Bytecode	Format	Description
`iload`	[1 byte op][1 byte X]	*Push local variable X*
`iload_0`	[1 byte op]	*Push local variable 0*
`iload_1`	[1 byte op]	*Push local variable 1*
...		
`istore`	[1 byte op] [1 byte X]	*Pop stack to local variable X*
`istore_0`	[1 byte op]	*Pop stack to local variable 0*
`istore_1`	[1 byte op]	*Pop stack to local variable 1*
...		
`bipush`	[1 byte op] [1 byte]	*Push int constant (-128...+127)*
`sipush`	[1 byte op] [2 bytes]	*Push int constant (-32768...+32767)*
`ldc`	[1 byte op] [1 byte idx]	*Push int constant from constant pool*
`iconst_0`	[1 byte op]	*Push int zero*
`iconst_1`	[1 byte op]	*Push int one*
...		

Storage Allocation

- Allocate space for local variables in a stack frame for that method.

  Keep track of next available location and record in symbol table.

  E.g. in above example: $x \rightarrow 1$, $y \rightarrow 2$.

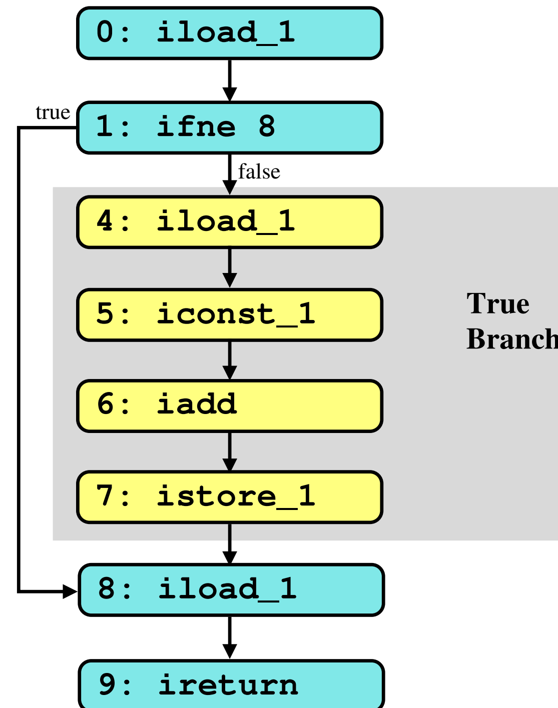
- Need to determine how much space each variable takes — mapping source language types to target data types.

- Need to determine scope/lifetime of variables, so storage can be reused.

- Can work out how much space is needed for a method at point of call.

Translating If-Statements

```
int f(int y) {
 if(y == 0) {
  y = y + 1;
 }
 return y;
}
```

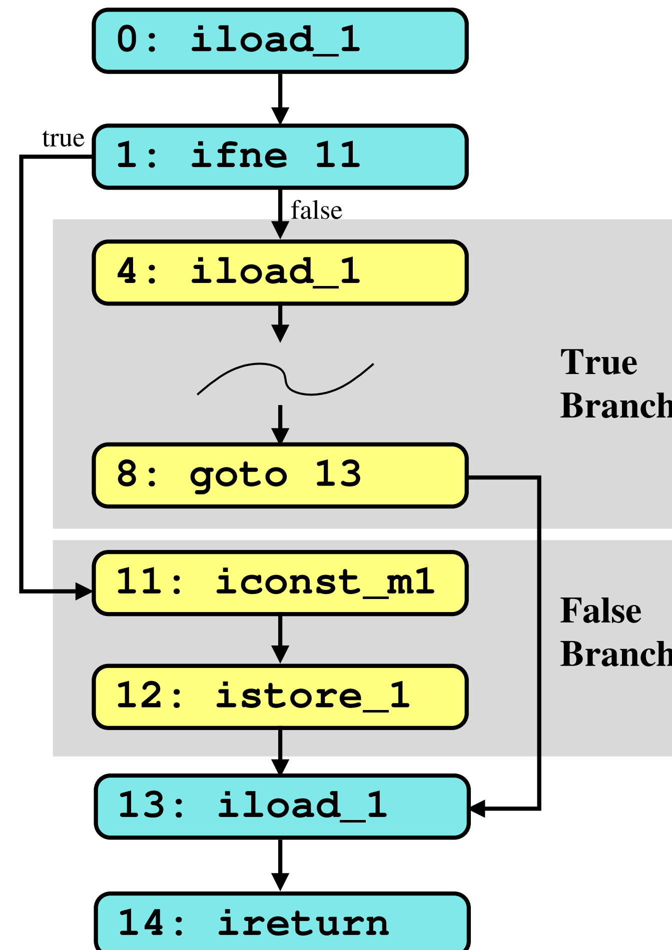


- In this case, no **else** branch — easy!
- But ... can't determine address for branch at line 1 until we've generated code for the true branch.

  Need to be able to insert instruction (or address) at an earlier location.

Translating If-Else-Conditionals

```
int f(int y) {
 if(y == 0) {
  y = y + 1;
 } else {
  y = -1;
 }
 return y;
}
```

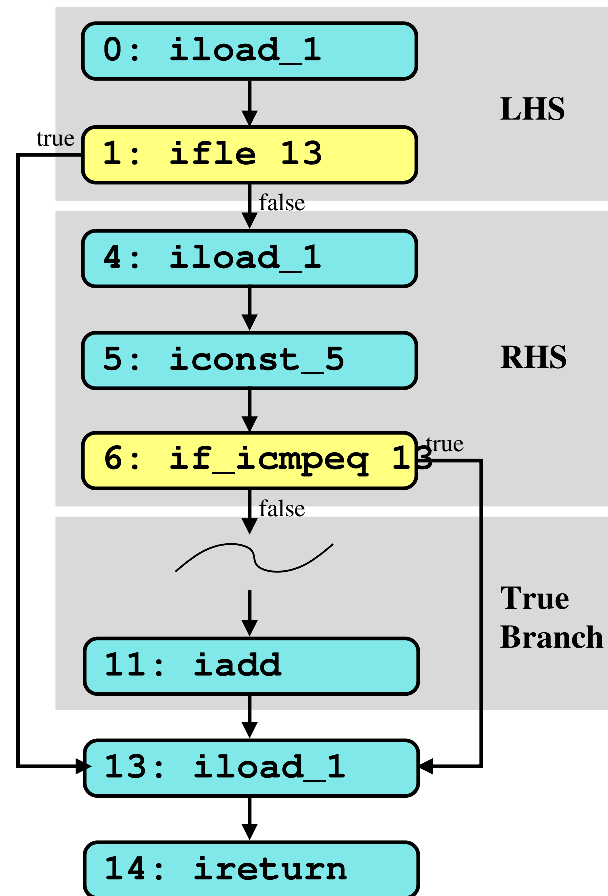


- The true branch **jumps over** the false branch!

Short Circuiting

- Logical connectives are translated using **short-circuiting**:

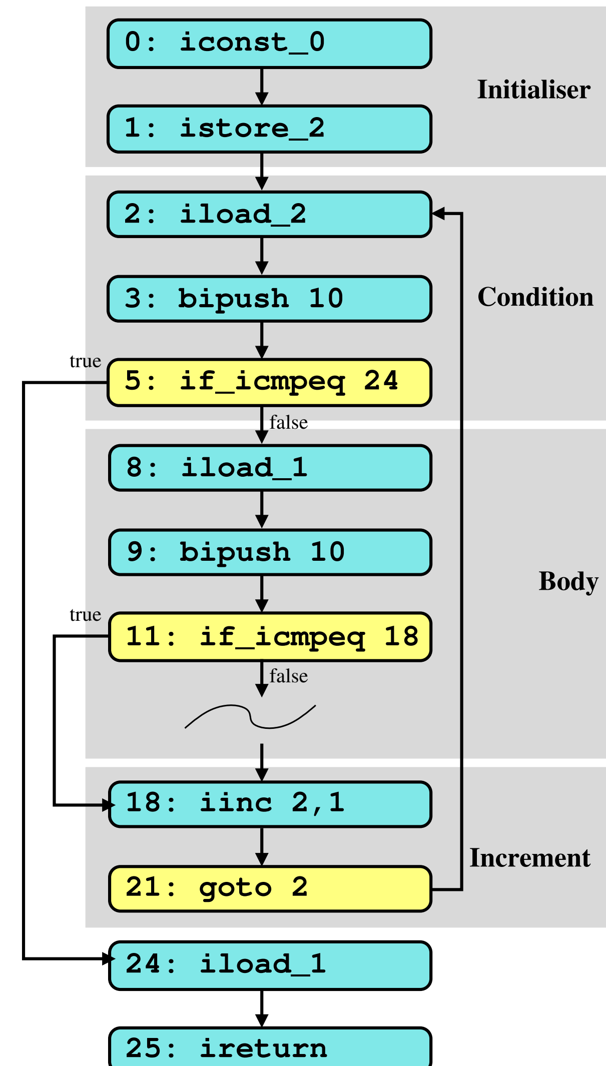
```
int f(int y) {
 if(y > 0 && y!=5) {
  y = y + 1;
 }
 return y;
}
```



- Here, right-hand expression **only executed if** left-hand gives true.

Translating Loops

```
int f(int y) {
 for(int i=0;i!=10;++i) {
  if(y==10) continue;
  y = y * 2;
 }
 return y;
}
```



Generating Branch Bytecodes

`goto`	[1 byte op][2 bytes offset] *Unconditional Branch (range -32768...+32767)*
`goto_w`	[1 byte op][4 bytes offset] *Unconditional Wide Branch (range $-2^{31}-1 \ldots 2^{31}$)*
`ifeq`	[1 byte op][2 bytes offset] *Branch if top two stack locations equal (range -32768...+32767)*

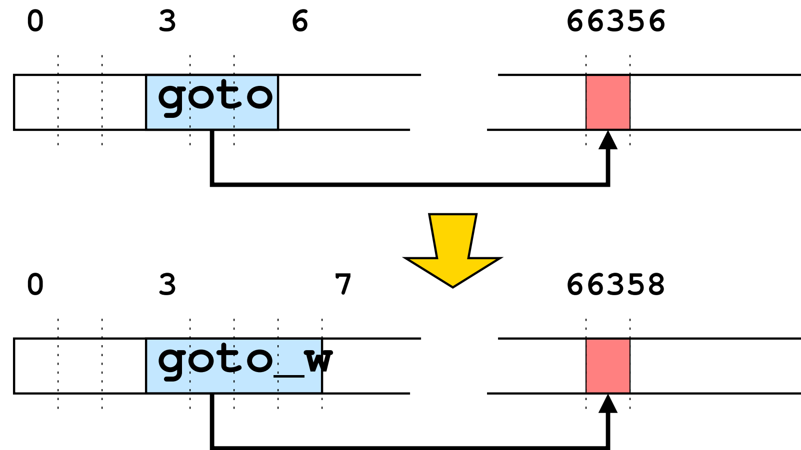
...

- Branch bytecodes use *relative addressing*, not *absolute addressing*
- Target address calculated by adding offset to current address:

```
void f(int):
  ...
  24: goto +35
  ...
  59: ...
```

Here, target address of **`goto`** bytecode is $(24+35)=59$

Calculating Branch Offsets



- Algorithm for calculating branch offsets:
  1. Generate all bytecodes, assuming branches take 3 bytes
  2. If branch exists which cannot reach target:

     Replace it with a *wide branch*:

     Update offsets of all branches (since they may have changed)
  3. Repeat step 2 until all branches can reach destination

- **Does this algorithm always terminate?**

  (need to consider padding of `tableswitch` + `lookupswitch`)

Generating Switch Bytecodes

- Two bytecodes for switch statements:

  `tableswitch` [op][padding][default][low][high][offsets]

  Padding: 0-3 zeroed bytes, so next byte word-aligned.
  Default: target address for default label
  Low: lowest value in case range
  High: Highest value in case range
  Offsets: Array of (high-low+1) Case Offsets

  `lookupswitch` [op][padding][default][npairs][pairs]

  padding: *0-3 zeroed bytes, so next byte word-aligned.*
  default: *target address for default label*
  npairs: *number of case value pairs*
  pairs: *array of pairs mapping case values to offsets*

Generating Switch Bytecodes (cont'd)

```
void f(int x) {
 int y;
 switch(x) {
  case 0:
   y = 1;
   break;
  case 1:
   y = 2;
  case 2:
   y = 3;
  default:
   y = -1;
 }
}
```

```
public void f(int);
   0:   iload_1
   1:   tableswitch
          default: 37
          low: 0
          high: 2
          offsets: +27, +32, +34
   28:  iconst_1
   29:  istore_2
   30:  goto    39
   33:  iconst_2
   34:  istore_2
   35:  iconst_3
   36:  istore_2
   37:  iconst_m1
   38:  istore_2
   39:  return
```

- Tableswitch is useful for contiguous case values
- **How many bytes of padding required here?**

Generating Switch Bytecodes (cont'd)

```
void f(int x) {
 int y;
 switch(x) {
  case 0:
   y = 1;
   break;
  case 12:
   y = 2;
  case 2046:
   y = 3;
  default:
   y = -1;
 }
}
```

```
public void f(int);
   0:   iload_1
   1:   lookupswitch
          default: 45
          npairs: 3
          pairs: 0->+35, 12->+40, 2046->+42
   36:  iconst_1
   37:  istore_2
   38:  goto    47
   41:  iconst_2
   42:  istore_2
   43:  iconst_3
   44:  istore_2
   45:  iconst_m1
   46:  istore_2
   47:  return
```

- Lookupswitch is useful for non-contiguous case values
- Notice that lookupswitch bytecode is much larger than before.

Generating Bytecode for WHILE

- Example translation of WHILE using `Jasm`:

```
void translate(Expr.IndexOf expr, Context context, List<Bytecode> bytecodes) {
   Attribute.Type attr = expr.getSource().attribute(Attribute.Type.class);
   // Translate source and index expressions
   translate(expr.getSource(),context,bytecodes);
   translate(expr.getIndex(),context,bytecodes);
   // Get value out of ArrayList
   JvmType.Function ft = new JvmType.Function(JvmTypes.JAVA_LANG_OBJECT, JvmTypes.INT);
   bytecodes.add(new Bytecode.Invoke(JAVA_UTIL_ARRAYLIST,"get",ft,InvokeMode.VIRTUAL));
   // unbox the element value on the stack
   Type elementType = ((Type.Array)attr.type).getElement();
   addReadConversion(elementType,bytecodes);
}
```

`https://github.com/Whiley/Jasm`