COMP3131/9102: Programming Languages and Compilers

Jingling Xue

School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia
`http://www.cse.unsw.edu.au/~cs3131`
`http://www.cse.unsw.edu.au/~cs9102`

Week 7: JVM (Two Lectures)

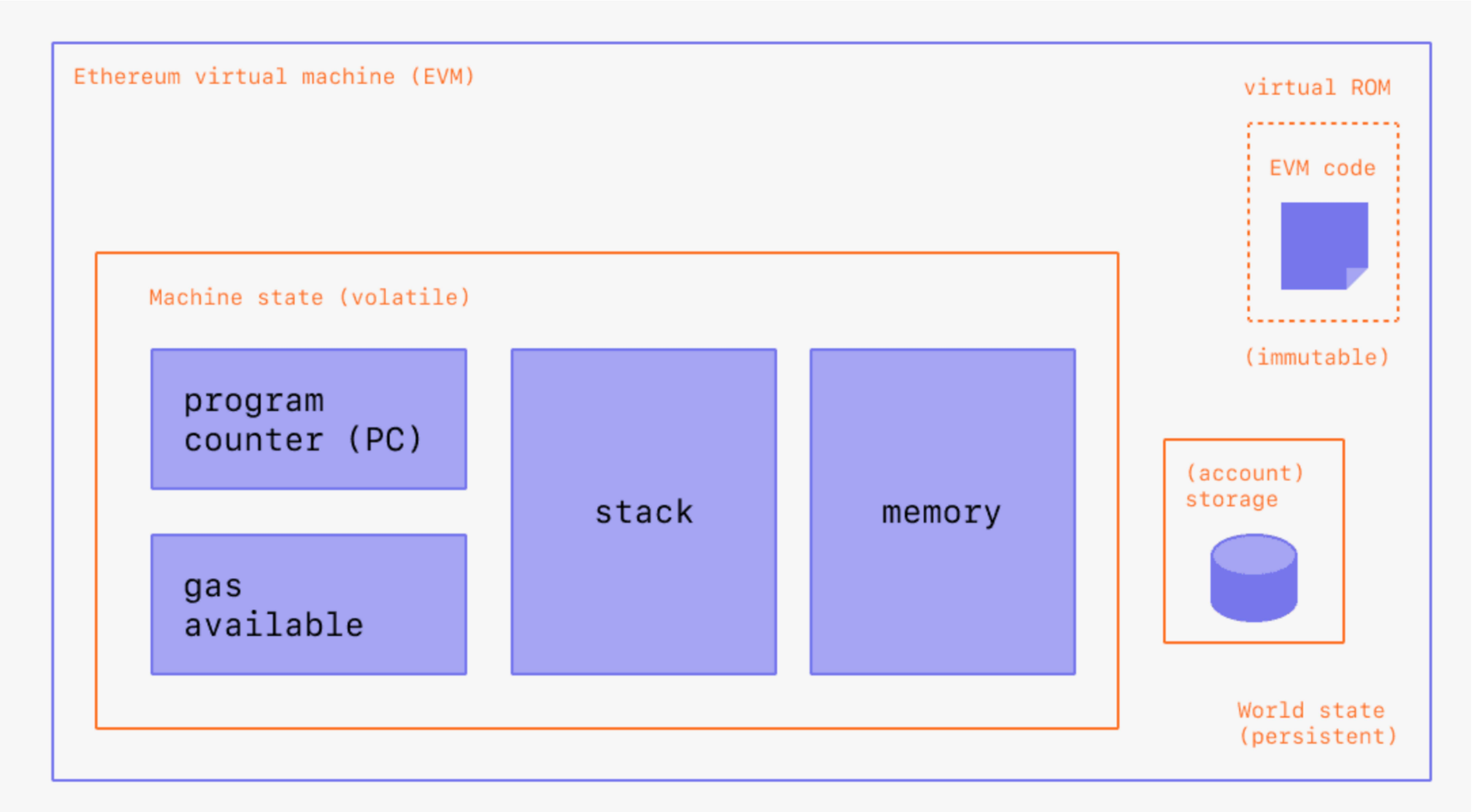
1st

1. Our code generation
2. JVM:
    - Data types
    - Operand stack
    - Local variable array (indices)

2nd

  - Instructions ($\Longleftrightarrow$ Jasmin instructions)
  - Parameter-passing ($\Longleftrightarrow$ Jasmin method invocations)

Ethereum Virtual Machine (EVM) for Blockchain



A good understanding about JVM and our last assignment will also help you understand EVM better.

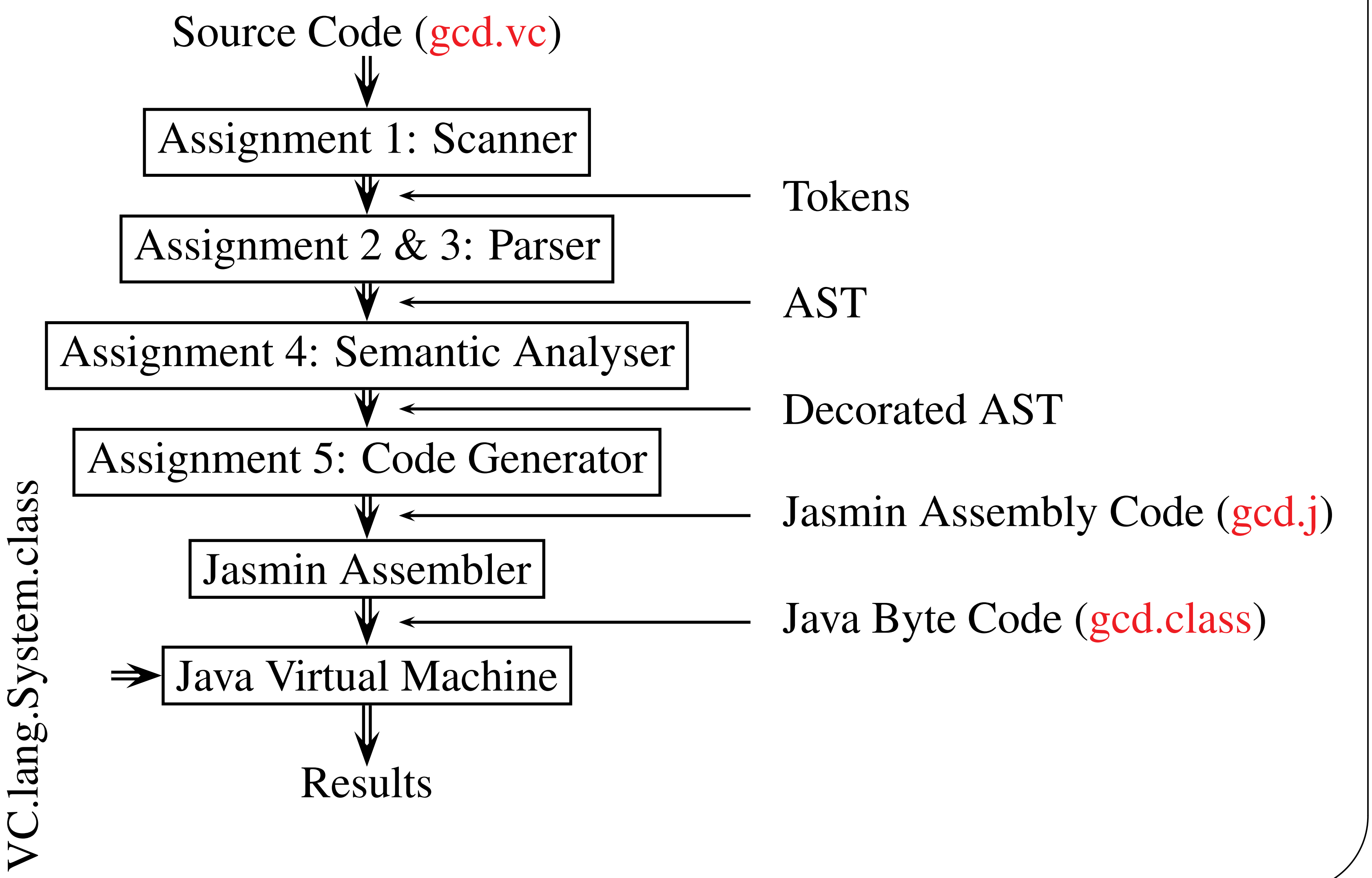
An Example

```
void whileInt() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

is compiled to

```
Method void whileInt()
   0    iconst_0
   1    istore_1 // i's index is 1
   2    goto 8
   5    iinc 1 1 // i++
   8    iload_1
   9    bipush 100
  11    if_icmplt 5
  14    return
```

The VC Compiler



Source Code (gcd.vc)

Assignment 1: Scanner

Tokens

Assignment 2 & 3: Parser

AST

Assignment 4: Semantic Analyser

Decorated AST

Assignment 5: Code Generator

Jasmin Assembly Code (gcd.j)

Jasmin Assembler

Java Byte Code (gcd.class)

VC.lang.System.class

Java Virtual Machine

Results

Standard Environment: Built-in Functions

```
/*
 * System.java
 */

package VC.lang;

import java.io.*;
import java.util.StringTokenizer;

public class System {

  private static BufferedReader reader =
        new BufferedReader(new InputStreamReader(java.lang.System.in));

  public final static int getInt() {
    try {
      java.lang.System.out.print("Enter an integer: ");
      String s = reader.readLine();
      StringTokenizer st = new StringTokenizer(s);
      int i = Integer.parseInt(st.nextToken());
      java.lang.System.out.println("You have entered " + i + ".");
      return i;
    } catch (java.io.IOException e) {
      java.lang.System.out.println("Caught IOException: " + e.getMessage());
      java.lang.System.exit(1);
      return -1;
    }
  }

  public final static void putBool(boolean b) {
    java.lang.System.out.print(b);
  }
```

```
public final static void putBoolLn(boolean b) {
  java.lang.System.out.println(b);
}

public final static void putInt(int i) {
  java.lang.System.out.print(i);
}

public final static void putIntLn(int i) {
  java.lang.System.out.println(i);
}

public final static float getFloat() {
  try {
    java.lang.System.out.print("Enter a float: ");
    String s = reader.readLine();
    StringTokenizer st = new StringTokenizer(s);
    float f = Float.parseFloat(st.nextToken());
    java.lang.System.out.println("You have entered " + f + ".");
    return f;
 } catch (java.io.IOException e) {
    java.lang.System.out.println("Caught IOException: " + e.getMessage());
    java.lang.System.exit(1);
    return -1.0F;
  }
}

public final static void putFloat(float f) {
  java.lang.System.out.print(f);
}

public final static void putFloatLn(float f) {
  java.lang.System.out.println(f);
}
```

```
  public final static void putString(String s) {
    java.lang.System.out.print(s);
  }

  public final static void putStringLn(String s) {
    java.lang.System.out.println(s);
  }

  public final static void putLn() {
    java.lang.System.out.println();
  }

}
```

References

- Jasmin home page: `http://jasmin.sourceforge.net/`
- Tim Lindholm and Frank Yellin, The Java Virtual Machine Specification, 2nd Edition, Addison-Wesley, 1999. (The entire book is available on-line; see the subject Resource Page.)
- Vill Venners, Inside the Java 2 Virtual Machine, 2nd Edition, McGraw-Hill, 1999.

  (Some chapters available on-line; see the subject Resource Page.)

Jasmin Assembly Language

- Sun has not defined an assembler format
- Jasmin is a Java assembler, which has been installed in the class account and can be invoked as follows:

```
% 3131
% jasmin gcd.j  --> the output is gcd.class
% java gcd
```

- Install from `http://jasmin.sourceforge.net/` on your own computer
- Read also the Jasmin User Guide there:

  `http://jasmin.sourceforge.net/guide.html`

- Jasmin page contains pointers to other assembly languages

Jasmin Assembly Language v.s Java Byte Code

- 1-to-1 correspondence
  - Operation codes (opcodes) represented by mnemonics
  - Name indices written in symbolic form
  - Local variables are encoded by indices (integers)
- Examples:

```
Jasmin Instructions            Java Byte Code
---------------------------------------------
iload                          0x60
bipush 20                      0x1614
getstatic Test.i               0xb2????
--------------------------------------------- where ????
is an index into the constant pool entry for Test.i
```

- Constant pool will be discussed in Week 8 but its understanding unnecessary for Assignment 5

gcd.java

```
// find the greatest common divisor of two integers

public class gcd {
  static int gcd(int a, int b) {
    if (b == 0)
      return a;
    else
    return gcd(b, a - (a/b) *b);
  }

  public static void main(String argv[]) {
    int i = 2;
    int j = 4;
    System.out.println(gcd(i, j));
  }
}
```

gcd.j

```
;; Produced by JasminVisitor (BCEL)
;; http://www.inf.fu-berlin.de/~dahm/BCEL/

.source gcd.java
.class public gcd
.super java/lang/Object


.method public <init>()V
.limit stack 1
.limit locals 1
.var 0 is this Lgcd; from Label0 to Label1

Label0:
.line 3
        aload_0
        invokespecial java/lang/Object/<init>()V
Label1:
        return

.end method

.method static gcd(II)I
.limit stack 4
.limit locals 2
.var 0 is a I from Label1 to Label2
.var 1 is b I from Label1 to Label2

Label1:
.line 5
        iload_1
        ifne Label0
.line 6
        iload_0
        ireturn
Label0:
.line 8
        iload_1
        iload_0
        iload_0
        iload_1
        idiv
        iload_1
        imul
        isub
        invokestatic gcd/gcd(II)I
Label2:
        ireturn

.end method

.method public static main([Ljava/lang/String;)V
.limit stack 3
.limit locals 3
.var 0 is argv [Ljava/lang/String; from Label0 to Label1
.var 1 is i I from Label2 to Label1
.var 2 is j I from Label4 to Label1

Label0:
.line 12
        iconst_2
        istore_1
Label2:
.line 13
        iconst_4
        istore_2
Label4:
.line 14
        getstatic java.lang.System.out Ljava/io/PrintStream;
        iload_1
        iload_2
        invokestatic gcd/gcd(II)I
        invokevirtual java/io/PrintStream/println(I)V
Label1:
.line 15
        return

.end method
```

Java Class File: gcd.class (Output of od -An -tx1 gcd.class)

```
ca fe ba be 00 03 00 2d 00 1e 0a 00 06 00 11 0a
00 05 00 12 09 00 13 00 14 0a 00 15 00 16 07 00
0b 07 00 17 01 00 06 3c 69 6e 69 74 3e 01 00 03
28 29 56 01 00 04 43 6f 64 65 01 00 0f 4c 69 6e
65 4e 75 6d 62 65 72 54 61 62 6c 65 01 00 03 67
63 64 01 00 05 28 49 49 29 49 01 00 04 6d 61 69
6e 01 00 16 28 5b 4c 6a 61 76 61 2f 6c 61 6e 67
2f 53 74 72 69 6e 67 3b 29 56 01 00 0a 53 6f 75
72 63 65 46 69 6c 65 01 00 08 67 63 64 2e 6a 61
76 61 0c 00 07 00 08 0c 00 0b 00 0c 07 00 18 0c
00 19 00 1a 07 00 1b 0c 00 1c 00 1d 01 00 10 6a
61 76 61 2f 6c 61 6e 67 2f 4f 62 6a 65 63 74 01
00 10 6a 61 76 61 2f 6c 61 6e 67 2f 53 79 73 74
65 6d 01 00 03 6f 75 74 01 00 15 4c 6a 61 76 61
2f 69 6f 2f 50 72 69 6e 74 53 74 72 65 61 6d 3b
01 00 13 6a 61 76 61 2f 69 6f 2f 50 72 69 6e 74
53 74 72 65 61 6d 01 00 07 70 72 69 6e 74 6c 6e
01 00 04 28 49 29 56 00 21 00 05 00 06 00 00 00
00 00 03 00 01 00 07 00 08 00 01 00 09 00 00 00
1d 00 01 00 01 00 00 00 05 2a b7 00 01 b1 00 00
00 01 00 0a 00 00 00 06 00 01 00 00 00 03 00 08
00 0b 00 0c 00 01 00 09 00 00 00 32 00 04 00 02
00 00 00 12 1b 9a 00 05 1a ac 1b 1a 1a 1b 6c 1b
68 64 b8 00 02 ac 00 00 00 01 00 0a 00 00 00 0e
00 03 00 00 00 05 00 04 00 06 00 06 00 08 00 09
00 0d 00 0e 00 01 00 09 00 00 00 34 00 03 00 03
00 00 00 10 05 3c 07 3d b2 00 03 1b 1c b8 00 02
b6 00 04 b1 00 00 00 01 00 0a 00 00 00 12 00 04
00 00 00 0c 00 02 00 0d 00 04 00 0e 00 0f 00 0f
00 01 00 0f 00 00 00 02 00 10
```

BCEL (Byte Code Engineering Library)

- Home page: `http://jakarta.apache.org/bcel/`
- Formerly known as JavaClass
- BCEL comes with a Jasmin disassembler, which has been installed in the class account and can be invoked as follows:

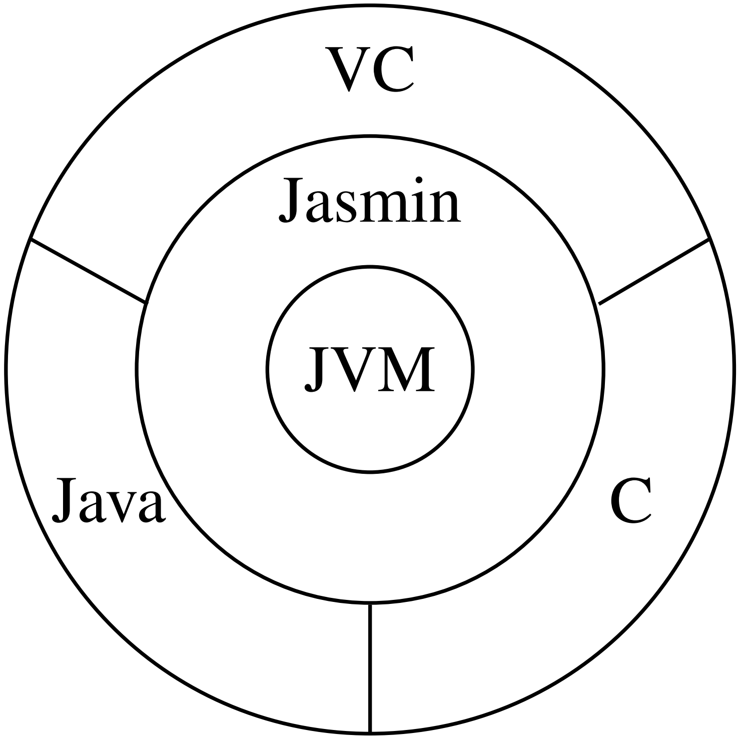
  ```
  % 3131
  % jasmind gcd.class --> this produces gcd.j
  ```

- `jasmind` is a shell command:

  ```
  #!/bin/csh
  #
  # jasmind - runs the Jasmin disassembler
  #
  # Usage:
  #     jasmind classname.class
  #
  setenv CLASSPATH ~cs3131/JavaTools/JavaClass
  exec java JasminVisitor $*
  ```

- Install from the BCEL home page on your own computer.

Multi-Level Machine Model



- There is a virtual machine and a language at each level
- Each level builds on the functionality of the level below and provides the functionality to the level above

Week 7 (1st Lecture): JVM

1. Our code generation $\surd$
2. JVM:
   - Data types
   - Operand stack
   - Local variable array (indices)

JVM Data Types

TYPE	RANGE	FIELD DESC
boolean	$\{0, 1\}$	Z
byte	8 bit signed 2's complement (-2^7 to $2^7 - 1$)	B
short	16 bit signed 2's complement (-2^{15} to $2^{15} - 1$)	S
int	32 bit signed 2's complement (-2^{31} to $2^{31} - 1$)	I
long	64 bit signed 2's complement (-2^{63} to $2^{63} - 1$)	L
char	16 bit unsigned Unicode (0 to $2^{16} - 1$)	C
float	32-bit IEEE 754 single-precision	F
double	64-bit IEEE 754 double-precision	D
reference	32 bit unsigned reference (0 to $2^{32} - 1$)	Slide 421
returnAddress	32 bit unsigned reference (0 to $2^{32} - 1$)	N/A

- All (except returnAddress) mapped 1-to-1 to Java's primitive types
- returnAddress used with jsr/jsr_w/ret for handling exceptions
- boolean, byte, char and short are all implemented as int, but arrays of these types may be stored in arrays of less than 32 bits

JVM Data Types (Cont’d)

TYPE	FIELD DESCRIPTOR
class reference	Lclass-name;
interface reference	Linterface-name;
array reference	$[[\cdots[$ component-type
void	V

- class and interface names are qualified names with ”.” replaced by ”/”
- The no. of $[$ is equal to the no. of dimensions of the array

Type	Field Descriptor
`Object`	`Ljava/lang/Object;`
`String`	`Ljava/lang/String;`
`String[]`	`[Ljava/lang/String;`
`int []`	`[I`
`float [][]`	`[[F`

- See $4.3.2, The JVM Spec for a formal definition

Boolean, Byte, Short and Char Represented as Int

```
public class IntTypes {
  public static void main(String argv[]) {
    boolean z = true;
    byte b = 1;
    short s = 2;
    char c = ’a’;
  }
}
.method public static main([Ljava/lang/String;)V
...

.line 3
        iconst_1
        istore_1
.line 4
        iconst_1
        istore_2
.line 5
        iconst_2
        istore_3
.line 6
        bipush 97
        istore 4
Label0:
.line 8
        return
.end method
```

An Example for Printing Data Type Descriptors

```
public class Desc {
  public static void main(String argv[]) {
    Object o = new Object();
    int [] i = new int[10];
    float [][] f = new float[10][10];
    String s1 = "Hello World!";
    String [] s2 = { "Hello", "World!"};

    System.out.println("Th class name of Object is: " + o.getClass());
    System.out.println("Th class name of int[] is: " + i.getClass());
    System.out.println("Th class name of float[][] is: " + f.getClass());
    System.out.println("Th class name of String: " + s1.getClass());
    System.out.println("Th class name of String[]: " + s2.getClass());

  }
}
```

Method Descriptors

- (ParameterType*) ReturnType
- Examples:

```
      Method Declaration        Method Descriptor
------------ -------------------------------------
int gcd(int i, int j)       (II)I
void main(String argv[])    ([Ljava/lang/String;)V
char foo(float f, String)   (FLjava/lang/String;)C
------------ -------------------------------------
```

- See $4.3.3, The JVM Spec for a formal definition

Operand Stack

- Accessed by pushing and popping values
  - storing operands and receiving the operations' results
  - passing arguments and receiving method results
  - This unified view is one of the main reasons why code generation for stack-based machines is easier than registers-based machines
- A new op stack is created every time a method is called
- Integral expression:

```
1 + 2 * 3 + 4
```

- Jasmin code (without being optimised):

```
iconst_1
iconst_2
iconst_3
imul
iadd
iconst_4
iadd
```

Local Variable Array

1. A new local variable array is created each time a method is called
2. Local variables addressed by indexing, starting from 0
3. Instance methods:
   - slot 0 given to this
   - Parameters (if any) given consecutive indices, starting from 1
   - The indices allocated to the other variables in any order
4. Class methods:
   - Parameters (if any) given consecutive indices, starting from 0
   - The indices allocated to the other variables in any order
5. One slot can hold a value of boolean, byte, char, short, int, float, reference and returnAdrress
6. One pair of slots can hold a value of long and double

Local Variable Indices: Class Methods

1. Class method:

```
public static void foo() {
  int i1 = 1;   // index 0
  int i2 = 2;   // index 1
  int i3 = 3;   // index 2
  int i = i1 + i2 * i3; // index 3
}
```

2. Jasmin code:

```
        iconst_1
        istore_0
        iconst_2
        istore_1
        iconst_3
        istore_2
        iload_0
        iload_1
        iload_2
        imul
        iadd
        istore_3
```

Local Variable Indices: Instance Methods

1. Instance method:

```
public void foo() {  // "this" given index 0
  int i1 = 1;   // index 1
  int i2 = 2;   // index 2
  int i3 = 3;   // index 3
  int i = i1 + i2 * i3; // index 4
}
```

2. Jasmin code:

```
        iconst_1
        istore_1
        iconst_2
        istore_2
        iconst_3
        istore_3
        iload_1
        iload_2
        iload_3
        imul
        iadd
        istore_4
```

Local Variable Indices: Double Word variables

1. The Long type:

```
public static void foo() {
  int i1 = 1;   // index 0
  long i2 = 2;   // indices 1 and 2
  int i3 = 3;   // index 3
  long i = i1 + i2 * i3; // indices 4 and 5
}
```

2. Jasmin code:

```
        iconst_1
        istore_0
        ldc2_w 2
        lstore_1
        iconst_3
        istore_3
        iload_0
        i2l
        lload_1
        iload_3
        i2l
        lmul
        ladd
        lstore 4
```

3. Accessing index 2 or 5 is disallowed

Week 7 (1st Lecture): JVM

1. Our code generation $\surd$
2. JVM:
   - Data types $\surd$
   - Operand stack $\surd$
   - Local variable array (indices) $\surd$
   - Instructions ($\Longleftrightarrow$ Jasmin instructions)
   - Parameter-passing ($\Longleftrightarrow$ Jasmin method invocations)

Reading (in Order of Increasing Importance)

- on-line JVM instructions
  `http://cs.au.dk/~mis/dOvs/jvmspec/ref-Java.html`
- Play around the tools mentioned in this lecture:
  - All available in the class account
  - Install them on your PC if you have one
- The JVM Spec Book
  - Chapter 3 (instructions)
  - Chapter 7 (more examples on compiling Java)
- “Inside the JVM” book (Chapter 5)

Next Class: Jasmin Assembly Language