The Java Virtual Machine (JVM)

Pat Morin
COMP 3002
Outline

• Topic 1
• Topic 2
  – Subtopic 2.1
  – Subtopic 2.2
• Topic 3
What is the JVM?

- The JVM is a specification of a computing machine
  - Instruction set
  - Primitive data types
  - Memory layout of primitive data types

- The JVM allows portable binary software
  - Authors need only write code for the JVM and it will run on any machine that has a JVM interpreter

- Hardware implementations of the JVM also exist
  - picoJava
Bytecodes: JVM Instructions

- A JVM instruction is a *opcode* followed by a variable number of operands
- An opcode is one byte
  - 225 different opcodes
- JVM byte codes are small compared to other instruction sets
  - On average, a JVM instruction is 1.8 bytes long
  - RISC instructions typically require 4 bytes
Distribution of Bytecode Lengths

- Source: Sun Microsystems
JVM Datatypes

- **Integers:**
  - byte (8 bits)
  - short (16 bits)
  - int (32 bits)
  - long (64 bits)

- **Floating point**
  - float (32 bits)
  - double (64 bits)

- **Others**
  - char (16 bits!)
  - reference (not defined)
  - returnAddress (not defined)
Parts of the JVM

- pc Register
  - Keeps track of the current instruction

- Stack
  - Stores stack frames

- Heap
  - Stores dynamically allocated memory (garbage collected)

- Method area
  - Stores program code (JVM instructions)

- Runtime constant pool
  - Stores constants and method/field references
What, no registers?

- The JVM has only one register, used to keep track of the program counter
- The JVM is a stack machine
  - All operations operate on the top few elements of the stack
- This is what allows for such short bytecodes
  - Eg. An integer division operation
    - in a 32 register RISC machine requires $32 \times 31 = 992$ different opcodes (one for each pair of registers)
    - in a stack machine requires only 1 opcode (it always operates on the top 2 elements)
The Jasmin Assembler

• Rather than manipulate bytecodes directly, we generate assembly language and use an assembler to assemble the byte codes

• The assembler we use is Jasmin
  – http://jasmin.sourceforge.net/
Hello World

```java
; public static void main(String args[]) {
;    System.out.println("Hello World!");
; }
.method public static main([Ljava/lang/String;)V
    .limit stack 50

    ; push System.out onto the stack
getstatic java/lang/System/out Ljava/io/PrintStream;
    ; push a string constant onto the stack
ldc "Hello World!"
    ; Call System.out.println
invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V

    return
.end method
```
Observe

- `.method` defines a new method
  - The parameters and return value are defined
- `.limit stack` defines the size (in 32 bit words) of the stack frame for this method
- `getstatic` loads a static (class) variable onto the stack
- `invokevirtual` calls a virtual (instance) method
- We return with an explicit return statement
Printing a Float

```java
public static void printFloat(float x) {
    System.out.println(Float.toString(x));
}
.method public static printFloat(F)V
    .limit stack 2
    .limit locals 2
    getstatic java/lang/System/out Ljava/io/PrintStream;
    fload 0
    invokestatic java/lang/Float/toString(F)Ljava/lang/String;
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    return
.end method
```
Local Variables

- `.locals` specifies the number of local variables
  - This includes function parameters
- Each local variable is a 32 bit quantity, indexed starting at 0
- Local variables are loaded onto and stored from the stack using `*load` and `*store` instructions
**load and store operations**

- The JVM has several load and store operations
  - `xload <index>`
    - `x` defines the type of operand (f, i, l, d, a ...)
    - index defines its index in the current stack frame

- The stack frame is indexed starting at 0
  - Starting with function arguments
  - Each index specifies a 4 byte quantity
    - byte, char, short, int, float, reference
    - “wide” data types are loaded in 2 steps
      - E.g., `dload_0 0 dload_1 1`

- store operations store the top element into the specified local variable
Operations

- The JVM includes many operations for arithmetic and logic
  - Add: iadd, ladd, fadd, dadd.
  - Subtract: isub, lsub, fsub, dsub.
  - Multiply: imul, lmul, fmul, dmul.
  - Divide: idiv, ldiv, fdiv, ddiv.
  - Remainder: irem, lrem, frem, drem.
  - Negate: ineg, lneg, fneg, dneg.
  - Comparison: dcmpg, dcmpl, fcmpg, fcmpl, lcmp.
  - and more ....

- They all operate on the top 1 or 2 stack elements and leave their result on the top of the stack
Labels and Jump Instructions

;;; return true (1) if arg1 >= arg2 and false (0) otherwise
.method public static cmpGE(FF)I
    .limit locals 4
    .limit stack 4
    fload 0
    fload 1
    fcmpl
    ifge true_label
    ldc 0
    goto done
true_label:
    ldc 1
done:
    ireturn
.end method
Summary

• The JVM is a stack-based virtual machine
  – Being stack-based allows for compact instruction encoding
  – Being a virtual machine makes it portable
  – Hardware implementations exist, but are not the fastest computers around

• We have not covered
  – Array instructions
  – Dynamic memory allocation
  – Exceptions
  – Interfaces
  – Type conversions
  – Threads