UNDER THE HOOD

By Bill Venners, JavaWorld

OCT 1, 1996 12:00 AM PST

Floating-point arithmetic

A look at the floating-point support of the Java virtual machine

‹ Page 2 of 2

Click here for the source code of Exposed Float.

Floating opcodes

The following table shows the opcodes that pop two floating-point values from the top of the stack, add them, and push the result. The type of the values is indicated by the opcode itself, and the result always has the same type as the numbers being added. No exceptions are thrown by these opcodes. Overflow results in a positive or negative infinity, and underflow results in a positive or negative zero.

Floating-point addition

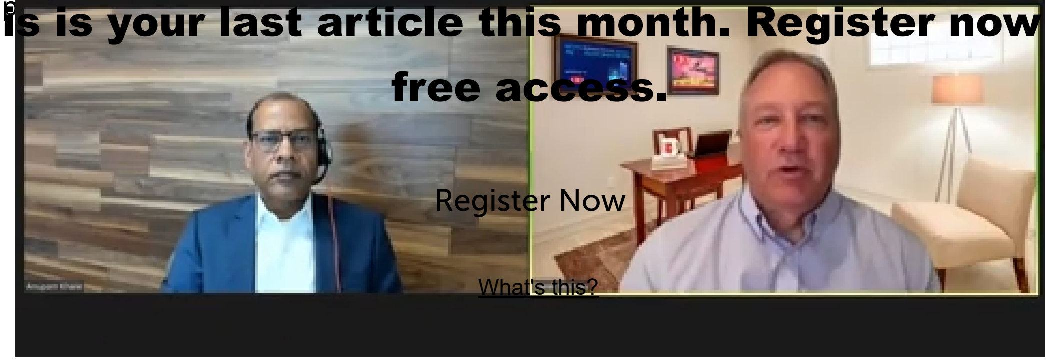
Opcode	Operand(s)	Description
`fadd`	(none)	pops two floats, adds them, and pushes the float result
`dadd`	(none)	pops two doubles, adds them, and pushes the double result

Subtraction is performed on floats and doubles via the following opcodes. Each opcode causes the top two values of the appropriate type to be popped off the stack. The topmost value is subtracted from the value just beneath the topmost value. The result is pushed back onto the stack. No exceptions are thrown by either of these opcodes.

Floating-point subtraction

Opcode	Operand(s)	Description
`fsub`	(none)	pops two floats, subtracts them, and pushes the float result
[illegible]	[illegible]	[illegible]lt

Multiplication of fl [illegible] s two values of the same type to be p [illegible] eing multiplied, is



Floating-point multiplication

Opcode	Operand(s)	Description
`fmul`	(none)	pops two floats, multiplies them, and pushes the float result
`dmul`	(none)	pops two doubles, multiplies them, and pushes the double result

The division experience is made available for floats and doubles by the following opcodes. The division opcodes cause the top two values of the appropriate type to be popped off the stack. The topmost value is divided by the value immediately beneath the topmost value. The result is pushed onto the stack. Floating-point division of a finite value by zero yields a positive or negative infinity. Floating-point division of zero by zero yields NaN. No exception is thrown as a result of any floating-point division.

Floating-point division

Opcode	Operand(s)	Description
`fdiv`	(none)	pops two floats, divides them, and pushes the float result
`ddiv`	(none)	pops two doubles, divides them, and pushes the double result

The remainder operation is accomplished via the following opcodes on floats and doubles. The following opcodes cause the top two values to be popped from the stack. The topmost value is divided by the value just beneath it, and the remainder of that division is pushed back onto the stack. Floating-point remainder of any value divided by zero yields a NaN result. No exception is thrown as a result of any floating-point division.

Floating-point remainder

Opcode	Operand(s)	Description
`frem`	(none)	pops two floats, divides them, and pushes the float remainder
`drem`	(none)	pops two doubles, divides them, and pushes the double remainder

The following opcodes perform arithmetic negation on floats and doubles. Negation opcodes pop the top value from the stack, negates it, and pushes the result.

Floating-point negation

Opcode	Operand(s)	Description
`fneg`	(none)	pops a float, negates it, and pushes the result
`dneg`	(none)	pops a double, negates it, and pushes the result

Circle of squares: A JVM simulation

The applet below demonstrates a Java virtual machine executing a sequence of bytecodes that perform floating-point arithmetic. The bytecode sequence in the simulation was generated by

`javac`

for the squareItForever() method of the class shown below:

What's this?

```
       class Struggle {
static void squareItForever() {
           float f = 2;
          while (true) {
              f = f * f;
              f = 0 - f;
                 }
               }
              }
```

The actual bytecodes generated by `javac` for squareItForever() are shown below:

```
                     fconst_2          // Push float constant 2.
         fstore_0          // Pop to local variable 0 (float f): float f = 2;
                fload_0           // Push local variable 0 (float f).
                fload_0           // Push local variable 0 (float f).
        fmul              // Pop top two floats, multiply, push float result.
          fstore_0          // Pop to local variable 0 (float f): f = f * f;
                     fconst_0          // Push float constant 0.
                fload_0           // Push local variable 0 (float f).
fsub              // Subtract top float from next to top float: imByte = (byte) imInt;
      fstore_0          // Pop result to local variable 0 (float f): f = 0 - f;
   goto 2            // Jump back to the first fload_0 instruction: while (true) {}
```

The squareItForever() method repeatedly squares a float value until it hits infinity. Each time the float is squared it is also negated. The float starts out as 2. It only takes seven iterations before infinity is reached, which isn't nearly as long as it takes in real life. The hex representation of the bits that make up the float are shown in the "hex value" column in the applet. The "value" column shows the number as humans are used to seeing it. This human-friendly value is generated by the Float.toString() method.

To drive the simulation, just press the "Step" button. Each press of the "Step" button will cause the JVM to execute one bytecode instruction. To start the simulation over, press the "Reset" button. The text area at the bottom of the applet describes the next instruction to be executed. Happy clicking.

Click here for the source code of Circle of Squares.

Bill Venners has been writing software professionally for 12 years.Based in Silicon Valley, he provides software consulting and training services under the name Artima Software Company. Over the years he has developed software for the consumer electronics, education, semiconductor, and life insurance industries. He has programmed in many languages on many platforms: assembly language on various microprocessors, C on Unix, C++ on Windows, Java on the Web. He is author of the book: Inside the Java Virtual Machine, published by McGraw-Hill.

Learn more about this topic

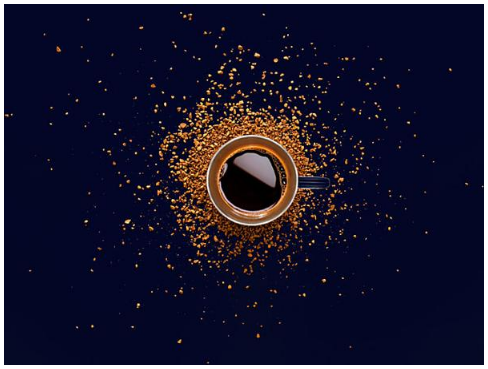
- "The Java Virtual Machine Specification" represents the official word from Sun. http://java.sun.com:80/doc/language_vm_specification.html (http://java.sun.com:80/doc/language_vm_specification.html)
- When it comes out, the book *The Java Virtual Machine Specification*, by Tim Lindholm and Frank Yellin (ISBN 0-201-63452-X), will be the definitive JVM reference. This book is part of *The Java Series* from Addison-Wesley http://www.aw.com/cp/lindholm-yellin.html (http://www.aw.com/cp/lindholm-yellin.html) http://www.aw.com/cp/javaseries.html (http://www.aw.com/cp/javaseries.html)

- The Java Language Specification discusses floating-point support in Java. This spec has just been published as a book as part of *The Java Series* from Addison-Wesley http://www.javasoft.com/doc/language_specification/ (http://www.javasoft.com/doc/language_specification/) http://www.aw.com/cp/javaseries.html (http://www.aw.com/cp/javaseries.html)
- Peter J. L. Wallis, ed. [1990] *Improving Floating-Point Programming,* John Wiley & Sons Ltd., ISBN 0 471 924377
- *Previous Under The Hood articles:*
- The lean, mean virtual machine (http://www.javaworld.com/javaworld/jw-06-1996/jw-06-vm.html) -- Gives an introduction to the Java virtual machine. Look here to see how the garbage collected heap fits in with the other parts of the JVM.
- The Java class file lifestyle (http://www.javaworld.com/javaworld/jw-07-1996/jw-07-classfile.html) -- Gives an overview of the Java class file, the file format into which all Java programs are compiled.
- Java's garbage-collected heap (http://www.javaworld.com/javaworld/jw-08-1996/jw-08-gc.html) -- Gives an overview of garbage collection in general and the garbage-collected heap of the Java virtual machine in particular.
- Bytecode basics (http://www.javaworld.com/javaworld/jw-09-1996/jw-09-bytecodes.html) -- Introduces the bytecodes of the JVM, and discusses primitive types, conversion operations, and stack operations.

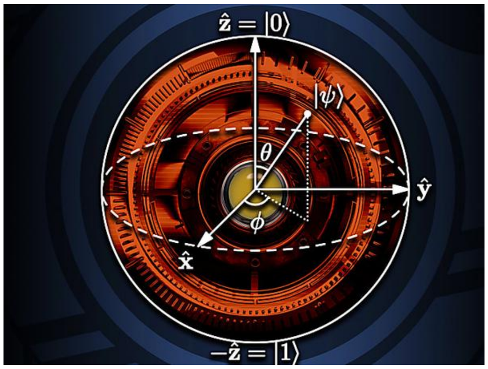
This story, "Floating-point arithmetic" was originally published by JavaWorld.

Page 2 of 2

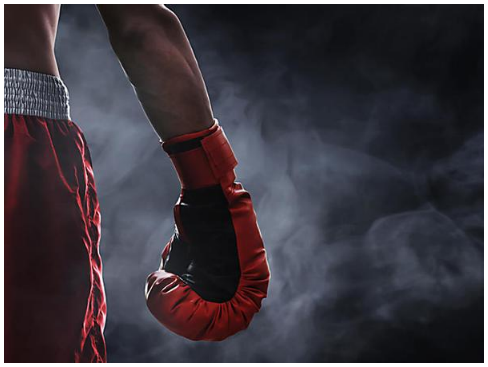
The 'intelligent edge' isn't always the smartest choice



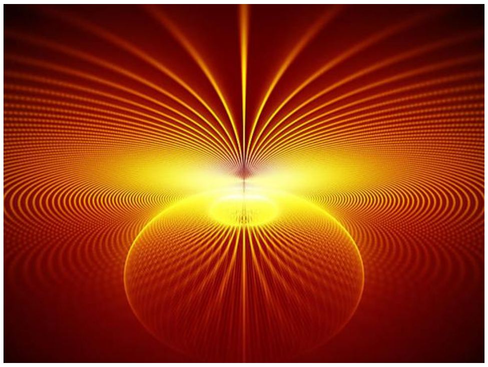
Kotlin queues up new compiler, WebAssembly back end



Amazon Braket: Get started with quantum computing



6 ways Alibaba Cloud challenges AWS, Azure, and GCP



Learn PyTorch: The best free online courses and tutorials

Servo browser engine adopted by Linux Foundation

4 nonprofits teaching disadvantaged students to code

Applying agile in POCs for emerging technologies

What to expect from AWS re:Invent 2020

Why developers should use graph databases

2 cloud architecture problems are still unsolved