JAVAWORLD

UNDER THE HOOD

By Bill Venners

HOW-TO

Floating-point arithmetic

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

JavaWorld | Oct 1, 1996 1:00 AM PT

Welcome to another installment of **Under The Hood**. This column aims to give Java developers a glimpse of the hidden beauty beneath their running Java programs. This month's column continues the discussion, begun last month, of the bytecode instruction set of the Java virtual machine (JVM). This article takes a look at floating-point arithmetic in the JVM, and covers the bytecodes that perform floating-point arithmetic operations. Subsequent articles will discuss other members of the bytecode family.

The main floating points

The JVM's floating-point support adheres to the IEEE-754 1985 floating-point standard. This standard defines the format of 32-bit and 64-bit floating-point numbers and defines the operations upon those numbers. In the JVM, floating-point arithmetic is performed on 32-bit floats and 64-bit doubles. For each bytecode that performs arithmetic on floats, there is a corresponding bytecode that performs the same operation on doubles.

A floating-point number has four parts -- a sign, a mantissa, a radix, and an exponent. The sign is either a 1 or -1. The mantissa, always a positive number, holds the significant digits of the floating-point number. The exponent indicates the positive or negative power of the radix that the mantissa and sign should be multiplied by. The four components are combined as follows to get the floating-point value:

$$\text{sign} * \text{mantissa} * \text{radix}^{\text{exponent}}$$

Floating-point numbers have multiple representations, because one can always multiply the mantissa of any floating-point number by some power of the radix and change the exponent to get the original number. For example, the number -5 can be represented equally by any of the following forms in radix 10:

Forms of -5

Sign	Mantissa	Radix ^{exponent}
-1	50	10^{-1}
-1	5	10^{0}
-1	0.5	10^{1}
-1	0.05	10^{2}

For each floating-point number there is one representation that is said to be *normalized.* A floating-point number is normalized if its mantissa is within the range defined by the following relation:

1/radix <= mantissa < 1

A normalized radix 10 floating-point number has its decimal point just to the left of the first non-zero digit in the mantissa. The normalized floating-point representation of -5 is $-1 * 0.5 * 10^{1}$. In other words, a normalized floating-point number's mantissa has no non-zero digits to the left of the decimal point and a non-zero digit just to the right of the decimal point. Any floating-point number that doesn't fit into this category is said to be *denormalized*. Note that the number zero has no normalized representation, because it has no non-zero digit to put just to the right of the decimal point. "Why be normalized?" is a common exclamation among zeros.

Floating-point numbers in the JVM use a radix of two. Floating-point numbers in the JVM, therefore, have the following form:

$$sign * mantissa * 2^{exponent}$$

The mantissa of a floating-point number in the JVM is expressed as a binary number. A normalized mantissa has its binary point (the base-two equivalent of a decimal point) just to the left of the most significant non-zero digit. Because the binary number system has just two digits -- zero and one -- the most significant digit of a normalized mantissa is always a one.

The most significant bit of a float or double is its sign bit. The mantissa occupies the 23 least significant bits of a float and the 52 least significant bits of a double. The exponent, 8 bits in a float and 11 bits in a double, sits between the sign and mantissa. The format of a float is shown below. The sign bit is shown as an "s," the exponent bits are shown as "e," and the mantissa bits are shown as "m":

Bit layout of Java float

s eeeeeeee mmmmmmmmmmmmmmmmmmmmmmm

A sign bit of zero indicates a positive number and a sign bit of one indicates a negative number. The mantissa is always interpreted as a positive base-two number. It is not a twos-complement number. If the sign bit is one, the floating-point value is negative, but the mantissa is still interpreted as a positive number that must be multiplied by -1.

The exponent field is interpreted in one of three ways. An exponent of all ones indicates the floating-point number has one of the special values of plus or minus infinity, or "not a number" (NaN). NaN is the result of certain operations, such as the division of zero by zero. An exponent of all zeros indicates a denormalized floating-point number. Any other exponent indicates a normalized floating-point number.

The mantissa contains one extra bit of precision beyond those that appear in the mantissa bits. The mantissa of a float, which occupies only 23 bits, has 24 bits of precision. The mantissa of a double, which occupies 52 bits, has 53 bits of precision. The most significant mantissa bit is predictable, and is therefore not included, because the exponent of floating-point numbers in the JVM indicates whether or not the number is normalized. If the exponent is all zeros, the floating-point number is denormalized and the most significant bit of the mantissa is known to be a zero. Otherwise, the floating-point number is normalized and the most significant bit of the mantissa is known to be one.

The JVM throws no exceptions as a result of any floating-point operations. Special values, such as positive and negative infinity or NaN, are returned as the result of suspicious operations such as division by zero. An exponent of all ones indicates a special floating-point value. An exponent of all ones with a mantissa whose bits are all zero indicates an infinity. The sign of the infinity is indicated by the sign bit. An exponent of all ones with any other mantissa is interpreted to mean "not a number" (NaN). The JVM always produces the same mantissa for NaN, which is all zeros except for the most significant mantissa bit that appears in the number. These values are shown for a float below:

Special float values

Value	Float bits (sign exponent mantissa)
+Infinity	0 11111111 00000000000000000000000
-Infinity	1 11111111 00000000000000000000000
NaN	1 11111111 10000000000000000000000

Exponents that are neither all ones nor all zeros indicate the power of two by which to multiply the normalized mantissa. The power of two can be determined by interpreting the exponent bits as a positive number, and then subtracting a bias from the positive number. For a float, the bias is 126. For a double, the bias is 1023. For example, an exponent field in a float of 00000001 yields a power of two by subtracting the bias (126) from the exponent field interpreted as a positive integer (1). The power of two, therefore, is 1 - 126, which is -125. This is the smallest possible power of two for a float. At the other extreme, an exponent field of 11111110 yields a power of two of (254 - 126) or 128. The number 128 is the largest power of two available to a float. Several examples of normalized floats are shown in the following table:

Normalized float values

Value	Float bits (sign exponent mantissa)	Unbiased exponent
Largest positive (finite) float	0 11111110 11111111111111111111111	128
Largest negative (finite) float	1 11111110 11111111111111111111111	128
Smallest normalized float	1 00000001 00000000000000000000000	-125
Pi	0 10000000 10010010000111111011011	2

An exponent of all zeros indicates the mantissa is denormalized, which means the unstated leading bit is a zero instead of a one. The power of two in this case is the same as the lowest power of two available to a normalized mantissa. For the float, this is -125. This means that normalized mantissas multiplied by two raised to the power of -125 have an exponent field of 00000001, while denormalized mantissas multiplied by two raised to the power of -125 have an exponent field of 00000000. The allowance for denormalized numbers at the bottom end of the range of exponents supports gradual underflow. If the lowest exponent was instead used to represent a normalized number, underflow to zero would occur for larger numbers. In

other words, leaving the lowest exponent for denormalized numbers allows smaller numbers to be represented. The smaller denormalized numbers have fewer bits of precision than normalized numbers, but this is preferable to underflowing to zero as soon as the exponent reaches its minimum normalized value.

Denormalized float values

Value	Float bits (sign exponent mantissa)
Smallest positive (non-zero) float	0 00000000 00000000000000000000001
Smallest negative (non-zero) float	1 00000000 00000000000000000000001
Largest denormalized float	1 00000000 11111111111111111111111
Positive zero	0 00000000 00000000000000000000000
Negative zero	1 00000000 00000000000000000000000

Exposed float

A Java float reveals its inner nature The applet below lets you play around with the floating-point format. The value of a float is displayed in several formats. The radix two scientific notation format shows the mantissa and exponent in base ten. Before being displayed, the actual mantissa is multiplied by 2^{24}, which yields an integral number, and the unbiased exponent is decremented by 24. Both the integral mantissa and exponent are then easily converted to base ten and displayed.