Exploring Binary

What Powers of Two Look Like Inside a Computer

By Rick Regan January 7th, 2009

A power of two, when expressed as a binary number, is easy to spot: it has one, and only one, 1 bit. For example, 1000, 10, and 0.001 are powers of two. Inside a computer, however, numbers are more generally represented in binary code, not as "pure" binary numbers. As a result, you may not be able to look at the binary representation of a number and tell at a glance whether it's a power of two or not; it depends on how it's encoded.

Numbers in computers are encoded in several forms, the two most common being ***binary integer*** and ***floating-point binary***. Powers of two are represented no differently, but because of their special relationship to binary notation, have a distinct look.

We'll start by reviewing the binary integer and floating-point binary representations, and then we'll see how powers of two are encoded in those forms.

Binary Integers

The standard way to represent integers is with two's complement notation. The two's complement representation of a nonnegative integer is faithful to its representation as a pure binary number, with some superfluous differences; it will not have commas or spaces and may have leading zeros added.

The two's complement representation of a negative integer is *not* faithful to its representation as a pure binary number. It has the equivalent of a minus sign, which is a 1 bit in the most significant or leftmost position, but the value in the rest of the field is not the pure binary representation of the integer. (That's all I'll say about it in this article, since negative numbers are not powers of two.)

Integers come in several sizes: 8 bits, 16 bits, 32 bits, and 64 bits are the common choices. The size determines the number of integers that can be represented. For example, a 32-bit integer can represent 2^{32} values.

An integer can be unsigned or signed. Unsigned integers represent only nonnegative values. A 32-bit unsigned integer can represent an integer between 0 and $2^{32} - 1$. Signed integers represent both negative and nonnegative values, with an equal number of each representable. A 32-bit signed integer can represent an integer between -2^{31} and $2^{31} - 1$.

Floating-Point Binary Numbers

The standard way to represent real numbers is with [IEEE 754 floating-point binary](). Floating-point representation is akin to scientific notation; it separates a number into a sign part, an exponent part, and a fraction part. This means that a floating point number is not stored as a pure binary number.

The two common forms of floating-point are single-precision, which is 32 bits, and double-precision, which is 64 bits. The forms differ by the size of the exponent and fraction fields; they are bigger in double-precision.

Single-Precision Floating-Point

Single-precision floating-point numbers are 32 bits long, and consist of three fields:

1. **Sign.** The sign field is 1 bit. It is 0 for positive numbers, 1 for negative numbers, and can be 0 or 1 for the number 0.
2. **Exponent.** The exponent field is 8 bits. It represents the exponent *n* to which 2 is raised, scaling the fraction by 2^n. The exponent is stored in biased form, which is a special encoding that stores positive and negative exponents as positive integers.
   To compute the exponent, interpret this field as an unsigned binary integer and subtract 127, giving a signed binary integer result. For example, a value of 10000011 (131) represents an exponent of 4, and a value of 01111101 (125) represents an exponent of -2. The values 0 and 255 have special meanings, and are not interpreted as exponents.

3. **Fraction.** The fraction field is 23 bits. If the number is normalized, the exponent field will be nonzero, and there will be an implicit 1 bit preceding the fraction. For example, 2^2 has an exponent field value of 10000001 (129), which means the exponent is 2. The number is interpreted as 1.0×2^2, but only the 0 is stored in the fraction.

If the number is denormalized (AKA denormal or subnormal), the exponent field will be zero, and there will *not* be an implicit 1 bit preceding the fraction. A denormalized number has an implicit starting exponent of -126, minus the position of the first 1 bit in the fraction field. For example, 2^{-128} has an exponent field value of zero, and a fraction field value of .01. The number is interpreted as 0.01 x 2^{-126}, which is 2^{-128}.

Double-Precision Floating-Point

Double-precision floating-point numbers are 64 bits long, but are otherwise similar to single-precision numbers. The only differences are in the exponent and fraction fields. The exponent field is 11 bits, has a bias of 1023, and gives the values 0 and 2047 special meanings. The fraction field is 52 bits.

Normalized and denormalized numbers are represented similarly as in single-precision floating-point. For example, 2^2, which is a normalized number, has an exponent value of 10000000001 (1025) and a fraction field value of 0. The number 2^{-1024}, which is a denormalized number, has an exponent value of zero and a fraction field value of .01.

Powers of Two in Integers

Powers of Two in Unsigned Integers

An n-bit unsigned integer can represent n powers of two; specifically, the nonnegative powers of two from 2^0 through 2^{n-1}. For example, a 32-bit unsigned integer can represent the 32 powers of two from 2^0 through 2^{31}:

Powers of Two in a 32-bit Unsigned Integer

Power of Two	32-Bit Unsigned Integer Representation
2^0	00000000000000000000000000000001
2^1	00000000000000000000000000000010
2^2	00000000000000000000000000000100
…	…
2^{29}	00100000000000000000000000000000

2^{30}	01000000000000000000000000000000
2^{31}	10000000000000000000000000000000

From the table you can deduce that an unsigned integer is a power of two if and only if it has exactly one 1 bit.

Powers of Two in Signed Integers

An n-bit signed integer can represent n-1 powers of two; specifically, the nonnegative powers of two from 2^0 through 2^{n-2}. For example, a 32-bit signed integer can represent the 31 powers of two from 2^0 through 2^{30}:

Powers of Two in a 32-bit Signed Integer

Power of Two	32-Bit Signed Integer Representation
2^0	00000000000000000000000000000001
2^1	00000000000000000000000000000010
2^2	00000000000000000000000000000100
…	…
2^{29}	00100000000000000000000000000000
2^{30}	01000000000000000000000000000000

The value 10000000000000000000000000000000, which is 2^{31} in an unsigned integer, is -2^{31} in a signed integer; thus, it's not a power of two in a signed integer.

From the table you can deduce that a signed integer is a power of two if and only if its sign bit is 0 and the remaining part of its binary representation has exactly one 1 bit.

Powers of Two in Floating-Point

Powers of Two in Single-Precision

Single-precision floating-point can represent 277 powers of two, from 2^{-149} through 2^{127}. Denormalized powers of two are those from 2^{-149} through 2^{-127}; normalized powers of two are those from 2^{-126} through 2^{127}. This table shows the range of powers of two covered and their encodings:

Powers of Two in a 32-Bit Floating-Point Number

Power of Two	32-Bit Floating-Point Representation		
	–	Exponent	Fraction
2^{-149}	0	00000000	00000000000000000000001
2^{-148}	0	00000000	00000000000000000000010
2^{-147}	0	00000000	00000000000000000000100
…	…	…	…
2^{-128}	0	00000000	01000000000000000000000
2^{-127}	0	00000000	10000000000000000000000
2^{-126}	0	00000001	00000000000000000000000
2^{-125}	0	00000010	00000000000000000000000
2^{-124}	0	00000011	00000000000000000000000
…	…	…	…
2^{-2}	0	01111101	00000000000000000000000
2^{-1}	0	01111110	00000000000000000000000
2^{0}	0	01111111	00000000000000000000000
2^{1}	0	10000000	00000000000000000000000
2^{2}	0	10000001	00000000000000000000000
2^{3}	0	10000010	00000000000000000000000
2^{4}	0	10000011	00000000000000000000000
…	…	…	…
2^{125}	0	11111100	00000000000000000000000
2^{126}	0	11111101	00000000000000000000000
2^{127}	0	11111110	00000000000000000000000

From the table you can deduce that a single-precision floating-point number is a power of two if and only if

- Its exponent field value is 0 and its fraction field has exactly one 1 bit
or
- Its exponent field value is between 1 and 254 and its fraction field is 0.

Powers of Two in Double-Precision

Double-precision floating-point can represent 2,098 powers of two, from 2^{-1074} through 2^{1023}. Denormalized powers of two are those from 2^{-1074} through 2^{-1023}; normalized powers of two are those from 2^{-1022} through 2^{1023}. This table shows the range of powers of two covered and their encodings:

Powers of Two in a 64-Bit Floating-Point Number

Power of Two	64-Bit Floating-Point Representation		
	–	Exponent	Fraction
2^{-1074}	0	00000000000	0000000000000000000000000000000000000000000000000001
2^{-1073}	0	00000000000	0000000000000000000000000000000000000000000000000010
2^{-1072}	0	00000000000	0000000000000000000000000000000000000000000000000100
…	…	…	…
2^{-1024}	0	00000000000	0100000000000000000000000000000000000000000000000000
2^{-1023}	0	00000000000	1000000000000000000000000000000000000000000000000000
2^{-1022}	0	00000000001	0000000000000000000000000000000000000000000000000000
2^{-1021}	0	00000000010	0000000000000000000000000000000000000000000000000000
2^{-1020}	0	00000000011	0000000000000000000000000000000000000000000000000000
…	…	…	…
2^{-2}	0	01111111101	0000000000000000000000000000000000000000000000000000
2^{-1}	0	01111111110	0000000000000000000000000000000000000000000000000000
2^{0}	0	01111111111	0000000000000000000000000000000000000000000000000000
2^{1}	0	10000000000	0000000000000000000000000000000000000000000000000000
2^{2}	0	10000000001	0000000000000000000000000000000000000000000000000000
2^{3}	0	10000000010	0000000000000000000000000000000000000000000000000000
2^{4}	0	10000000011	0000000000000000000000000000000000000000000000000000
…	…	…	…
2^{1021}	0	11111111100	0000000000000000000000000000000000000000000000000000

2^{1022}	0	11111111101	0000000000000000000000000000000000000000000000000000
2^{1023}	0	11111111110	0000000000000000000000000000000000000000000000000000

From the table you can deduce that a double-precision floating-point number is a power of two if and only if

- Its exponent field value is 0 and its fraction field has exactly one 1 bit
  or
- Its exponent field value is between 1 and 2046 and its fraction field is 0.

Endianness

Endianness is a property of computers that specifies how bytes within the binary encoding of a number are ordered in memory. There are two orderings, big-endian and little-endian. Big-endian can be viewed as the ordering in the tables above; little-endian can be viewed as the reverse. For example, in single-precision floating-point, 2^{-2} is 00111110100000000000000000000000 in big-endian and 00000000000000001000000000111110 in little-endian. Endianness applies similarly to double-precision floating-point and all integers greater than 8 bits long.

If you're working on a little-endian machine — which you are if you're on an Intel-based PC — you'll have to reverse the bytes of the number representation to have it match the layout in the tables.

Summary

You can tell whether a number in a computer is a power of two by looking at its binary encoding. Unsigned integers are easiest to check; just count 1 bits. Signed integers are almost as easy to check; check that the number is nonnegative and then count 1 bits. Numbers in floating-point representation are harder to check. You must parse the fields, taking into account endianness, and then interpret each field according to how it's defined. In other words, a simple count of 1 bits is insufficient.

Try it Out

I wrote a C program to print the fields of a double-precision floating-point-number. You can use that program to explore the representation of powers of two further.

EB

Related

- Displaying IEEE Doubles in Binary Scientific Notation
- Displaying the Raw Fields of a Floating-Point Number
- A Simple C Program That Prints 2,098 Powers of Two
- Ten Ways to Check if an Integer Is a Power Of Two in C
- How to Check If a Number Is a Power of Two

Get articles by RSS (What Is RSS?)

Get articles by e-mail

Numbers in computers / Binary integers, Exponents, Floating-point

3 comments

1. **krupal**

   September 8, 2014 at 11:25 am

   Thanks a lot…This solved my all issues in understanding floating point numbers.

2. **AndyF**

   April 8, 2017 at 7:02 pm

   Hi, there's a small typo in this sentence from your above article: "A 32-bit signed integer can represent an integer between -2^-31 and 2^31 – 1."
   It should be – (minus) 2 raised to the power of positive 31, not to the power of negative 31. (Apologies, I can't reproduce the superscript in this comment, so I used the '^' character to indicate "to the power of".)

3. **Rick Regan**

   April 8, 2017 at 10:15 pm

   @AndyF,

   Thanks for spotting that — I just fixed it. It was there for 8 years!

Comments are closed.