Exploring Binary

Number of Bits in a Decimal Integer

By Rick Regan December 13th, 2012

Every integer has an equivalent representation in decimal and binary. Except for 0 and 1, the binary representation of an integer has more digits than its decimal counterpart. To find the number of binary digits (bits) corresponding to any given decimal integer, you could convert the decimal number to binary and count the bits. For example, the two-digit decimal integer 29 converts to the five-digit binary integer 11101. But there's a way to compute the number of bits directly, without the conversion.

Sometimes you want to know, not how many bits are required for a *specific* integer, but how many are required for a *d*-digit integer — a *range* of integers. A range of integers has a range of bit counts. For example, four-digit decimal integers require between 10 and 14 bits. For any *d*-digit range, you might want to know its minimum, maximum, or average number of bits. Those values can be computed directly as well.

In this article, I will show you those calculations. I will be discussing *pure* binary and decimal, not computer encodings like two's complement, fixed-point, floating-point, or BCD. All of the discussion assumes positive integers, although it applies to negative integers if you temporarily ignore their minus signs. 0 is a special case not covered by the formulas, but obviously it has only 1 bit.

(I use the terms *decimal integer* and *binary integer* when I really mean "an integer expressed in decimal numerals" and "an integer expressed in binary numerals". An integer is an integer, independent of its base.)

Number of Bits in a Specific Decimal Integer

A positive integer n has b bits when $2^{b-1} \le n \le 2^b - 1$. For example:

- 29 has **5** bits because $16 \le 29 \le 31$, or $2^4 \le 29 \le 2^5 - 1$
- 123 has **7** bits because $64 \le 123 \le 127$, or $2^6 \le 123 \le 2^7 - 1$
- 967 has **10** bits because $512 \le 967 \le 1023$, or $2^9 \le 967 \le 2^{10} - 1$

For larger numbers, you could consult a table of powers of two to find the consecutive powers that contain your number.

To see why this works, think of the binary representations of the integers 2^4 through $2^5 - 1$, for example. They are 10000 through 11111, all possible 5-bit values.

Using Logarithms

The above method can be stated another way: the number of bits is the exponent of the smallest power of two greater than your number. You can state that mathematically as:

$$b_{spec} = \lfloor \log_2(n) \rfloor + 1$$

That formula has three parts:

- **$\log_2(n)$** means the ***logarithm in base*** 2 of n, which is the exponent to which 2 is raised to get n. For example, $\log_2(123) \approx 6.9425145$. The presence of a fractional part means n is between powers of two.
- **$\lfloor x \rfloor$** is the ***floor*** of x, which is the integer part of x. For example, $\lfloor 6.9425145 \rfloor = 6$. You can think of $\lfloor \log_2(n) \rfloor$ as the exponent of the highest power of two in the binary representation of n.
- **+ 1** takes the exponent to the next higher power of two. You can think of this step as accounting for the 2^0th place of your binary number, which then gives you its total number of bits. For our example, that's 6 + 1 = 7.

You might be tempted to use the ceiling function — $\lceil x \rceil$, which is the smallest integer greater than or equal to x — to compute the number of bits as such:

$b_{\text{spec}} = \lceil \log_2(n) \rceil$

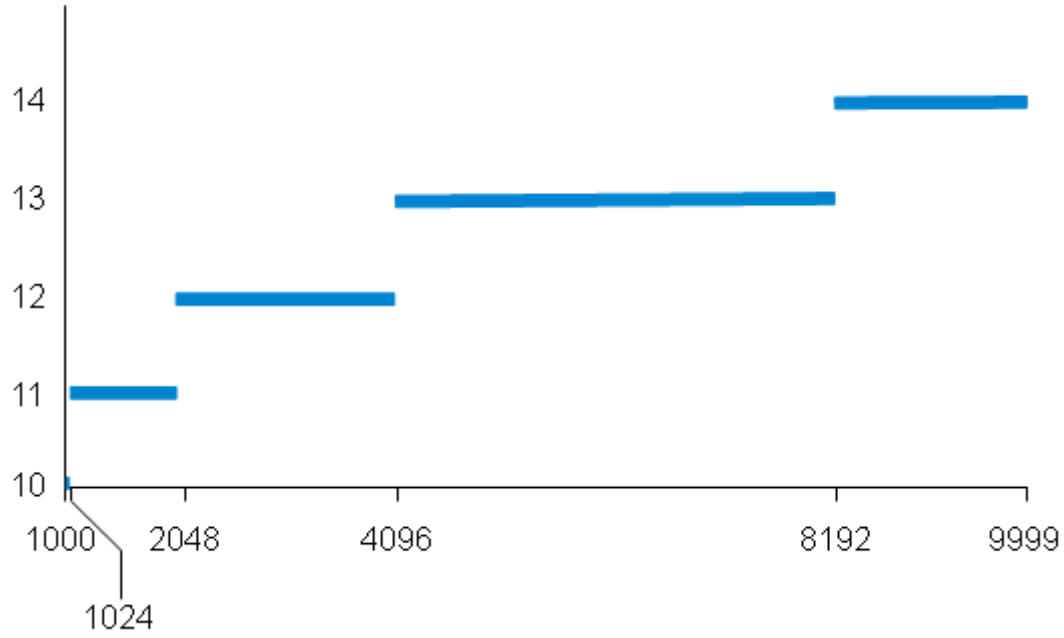
However, this fails when n is a power of two.

Number of Bits in a d-Digit Decimal Integer

A positive integer n has d decimal digits when $10^{d-1} \le n \le 10^d - 1$. How many bits do numbers in this range require? It varies. For example, consider four-digit decimal integers. Using the above formula you'll see that the smallest four-digit number, 1000, requires 10 bits, and the largest four-digit number, 9999, requires 14 bits. The number of bits varies between those extremes. For example, 1344 requires 11 bits, 2527 requires 12 bits, and 5019 requires 13 bits. Why does this occur? Because that single power of ten range spans all or part of five consecutive power-of-two ranges. Here's how the examples look from that viewpoint:

- 1000 has **10** bits because $512 \le 1000 \le 1023$, or $2^9 \le 1000 \le 2^{10} - 1$
- 1344 has **11** bits because $1024 \le 1344 \le 2047$, or $2^{10} \le 1344 \le 2^{11} - 1$
- 2527 has **12** bits because $2048 \le 2527 \le 4095$, or $2^{11} \le 2527 \le 2^{12} - 1$
- 5019 has **13** bits because $4096 \le 5019 \le 8191$, or $2^{12} \le 5019 \le 2^{13} - 1$
- 9999 has **14** bits because $8192 \le 9999 \le 16383$, or $2^{13} \le 9999 \le 2^{14} - 1$

This diagram shows the ranges:



Number of Bits In Four-Digit Decimal Integers

Minimum Number of Bits in a d-Digit Integer

The minimum number of bits required for a d-digit integer is computed simply by using the specific number formula on the minimum d-digit value:

$$b_{min} = \lfloor \log_2(10^{d-1}) \rfloor + 1$$

We can make this a more efficient computation by using the logarithmic identity $\log_a(x^y) = y \cdot \log_a(x)$:

$$b_{min} = \lfloor \log_2(10^{d-1}) \rfloor + 1 = \mathbf{\lfloor (d-1) \cdot \log_2(10) \rfloor + 1}$$

In this form, we take the logarithm of a small constant instead of a large variable.

Since we are dealing with powers of ten we can use the ceiling function here (as long as $d > 1$); there is no positive power of ten that is also a power of two. Here's the equivalent formula:

$$b_{min} = \lceil \log_2(10^{d-1}) \rceil = \mathbf{\lceil (d-1) \cdot \log_2(10) \rceil}$$

Maximum Number of Bits in a d-Digit Integer

The maximum number of bits required for a d-digit integer is computed simply by using the specific number formula on the maximum d-digit value:

$$b_{max} = \lfloor \log_2(10^d - 1) \rfloor + 1$$

We can't make the same simplification as for the minimum value, at least not on the face of it. But notice that $\lfloor \log_2(10^d - 1) \rfloor = \lfloor \log_2(10^d) \rfloor$, since a power of ten and that power of ten minus one are both in the same power of two range. (A power of ten minus one cannot be a power of two — it's odd). This allows us to use this more computationally efficient formula to the same effect:

$$b_{max} = \lfloor \log_2(10^d) \rfloor + 1 = \mathbf{\lfloor d \cdot \log_2(10) \rfloor + 1}$$

And again, for $d > 1$, we can use the ceiling function instead:

$$b_{max} = \mathbf{\lceil d \cdot \log_2(10) \rceil}$$

Average Number of Bits in a d-Digit Integer

The average number of bits required for a d-digit integer is the total number of bits required to represent all d-digit integers divided by the number of d-digit integers. For our example, the average is

$$b_{avg} = (24 \cdot 10 + 1024 \cdot 11 + 2048 \cdot 12 + 4096 \cdot 13 + 1808 \cdot 14)/9000 \approx 12.74$$

Computing The Formulas

Of the above formulas, these two — in these forms — are the most commonly used:

- $b_{spec} = \lfloor \log_2(n) \rfloor + 1$
- $b_{max} = \lceil d \cdot \log_2(10) \rceil$

How do you compute them?

First we have to figure out how to take the base-2 logarithm of a number. Many programming environments do not have a base-2 logarithm function. You can deal with that by doing a change of base:

$\log_2(n) = \log_a(n)/\log_a(2)$

'*a*' could represent any base; for example, base *e*, the natural logarithm. For simplicity, I'll drop the '*a*' and refer to the logarithm function generically as log(n):

$\log_2(n) = \log(n)/\log(2)$

Computing b_{spec}

Let's restate the formula in the form you would most likely enter it in a computer:

```
b_spec = floor(log(n)/log(2)) + 1
```

For example, the decimal integer 1,997,443,410 has floor(log(1997443410)/log(2)) + 1 = 31 bits.

You have to be careful when computing logarithms; floating-point inaccuracies could cause an incorrect result, particularly at power-of-two boundaries.

(If you are using a language like C you could avoid change of base — and improve performance and accuracy — by computing the floor of the base-2 logarithm using "Bit Twiddling Hacks".)

For very large numbers, you'll need arbitrary-precision arithmetic. I use PARI/GP, for example.

Computing b_{max}

Let's restate the formula for computer evaluation:

```
b_max = ceil(d*(log(10)/log(2)))
```

(The parentheses around the division are unnecessary; I just like thinking of log(10)/log(2) as a separate constant.)

log(10)/log(2), to 17 digits, is 3.3219280948873623. People commonly round this to 3.32, leading to this simple formula:

b_{max} = ceil(d*3.32)

For example, a 16-digit decimal integer requires up to ceil(16*3.32) = 54 bits.

You have to be careful with this though; for example, ceil(25*3.32) = 83 (since 25*3.32 = 83), but 25-digit integers require 84 bits. You need to specify the constant with more precision to get the correct result. (Here's one place where this error is made.)

EB

Related

- Number of Decimal Digits In a Binary Fraction

Get articles by RSS (What Is RSS?)

Get articles by e-mail

Binary numbers / Binary integers, Code, Convert to binary, Floating-point

13 comments

1. **Willie**

   January 23, 2014 at 3:22 am

   https://www.exploringbinary.com/number-of-bits-in-a-decimal-integer/

   Number of Bits in a Specific Decimal Integer
   A positive integer n has b bits when $2^{b-1} \le n \le 2^b - 1$. For example:

29 has 5 bits because 16 ≤ 29 ≤ 31, or 2^4 ≤ 29 ≤ 2^5 – 1

I think there is something wrong with the counting of bits on your site.
Yes, I agreed that 29 is between 2^4 and 2^5 – 1, however 2^5 should be 6 bits instead of 5 bits.
2^5 2^4 2^3 2^2 2^1 2^0
6 5 4 3 2 1 bits

---

2. **Rick Regan**

January 23, 2014 at 10:20 am

@Willie,

Yes, 2^5 is 6 bits and my formula gives that. Can you show me specifically what you calculated?

---

3. **Roy**

April 7, 2014 at 1:35 pm

Nice,I like it. Sometimes blogs are better than books!

I heard in class that an integer of size n takes O(log n) bits. Why is that? Shouldn't this be (floor(log n) + 1) bits ? Some people just say 'an integer is represented in a computer by log n bits.' Why is that ?

---

4. **Rick Regan**

April 7, 2014 at 3:44 pm

@Roy,

Thanks for the feedback.

"Big O" notation just gives you the overall trend of a particular function, not its exact value. The "floor" and "+1" parts are insignificant in that regard. When people just say 'is represented in a computer by log n bits' they are implicitly using order notation.

(BTW, I assume you meant “an integer n” and not “an integer of size n”.)

---

5. **b_rad**

   April 30, 2014 at 11:41 pm

   If you’re tempted to use the ceiling function the right way to calculate the number of bits needed to represent a positive integer n, use it like this:

   bspec = ⌈log2(n+1)⌉

---

6. **Rick Regan**

   May 2, 2014 at 11:29 am

   @b_rad,

   Yes, you are correct. That is used commonly too. It’s easy to show why it works, but I don’t think it is as intuitive (changing the argument to log() vs. changing the result of log()).

---

7. **P.Preethi**

   June 19, 2014 at 6:34 am

   Nice post

---

8. **Wasim Thabraze**

   November 5, 2014 at 9:05 am

   Thanks Rick! That was an amazing post. 🙂

---

9. **Aravind**

   December 21, 2015 at 4:45 pm

Thank you for your information sir. Can I know the formula for calculating the number of digits for n-bits? Could you mail me if possible.
Thank you sir.

---

10. **Rick Regan**

December 21, 2015 at 9:17 pm

@Aravind,

Check out the companion article: https://www.exploringbinary.com/number-of-decimal-digits-in-a-binary-integer/ .

---

11. **Vachaspati Pandey**

April 18, 2016 at 8:21 pm

Sir,
It was a very informative post. I came across it through google search for minimum bits required for representing a decimal fraction. I am still searching for the answer. Any suggestion please.

---

12. **Rick Regan**

April 18, 2016 at 8:27 pm

@Vachaspati,

Please explain what you are looking for (an example would help).

---

13. **Carrie**

September 21, 2016 at 1:53 pm

Homework stinks 😕 Thx for the help! c:

Comments are closed.