Exploring Binary

# Pi and e In Binary

By Rick Regan May 18th, 2011

Some people are curious about the binary representations of the mathematical constants pi and e. Mathematically, they're like every other irrational number — infinite strings of 0s and 1s (with no discernible pattern). In a computer, they're finite, making them only approximations to their true values. I will show you what their approximations look like in five different levels of binary floating-point precision.

$\pi$ = 11.001001000011111101101010100010001000011011...

e = 10.101101111110000101010001011000101000101101...

*The first 43 bits of pi and e*

## Binary Floating-Point Formats

In binary floating-point, infinitely precise values are rounded to finite precision. Here's how rounding works in five different levels of precision:

- In *half-precision*, values are rounded to 11 significant bits.
- In *single-precision*, values are rounded to 24 significant bits.
- In *double-precision*, values are rounded to 53 significant bits.
- In *extended-precision*, values are rounded to 64 significant bits.
- In *quadruple-precision*, values are rounded to 113 significant bits.

The rounding rule used most often in practice is round-to-nearest, round-half-to-even; that's the rule I will use. For pi and *e*, there are no "half to even" cases, since their binary expansions are infinite. This makes the rounding rule simple: if the rounding bit is 0, round down; if the rounding bit is 1, round up.

I will show the correctly rounded approximations of pi and *e* in these five formats.

# Pi (π)

Here are the first 50 decimal digits of pi:

3.14159265358979323846264338327950288419716939937510...

Here are the first 128 bits of pi:

```
11.00100100001111110110101010001000100001011010001100001000110100110001001100
0011001100010100010111000000011011100000111001101000 1...
```

Here are the 128 bits again, with the rounding bit for each level of precision highlighted (bits 12, 25, 54, 65, and 114):

```
11.00100100001111110110101010001000100001011010001100001000110100110001001100
0011001100010100010111000000011011100000111001101000 1...
```

Here are the correctly rounded values of pi in each of the five levels of precision, shown in normalized binary scientific notation and as hexadecimal floating-point constants:

## Half-Precision

$$\begin{aligned} pi &= 1.1001001 \times 2^{1} \\ &= 0x1.92p+1 \end{aligned}$$

This equals 3.140625 in decimal (all binary floating-point numbers have exact decimal representations), which approximates pi accurately to about 4 decimal digits.

(You can verify this conversion by hand, by adding the powers of two corresponding to the positions of the 1 bits: $11.001001 = 2^1 + 2^0 + 2^{-3} + 2^{-6} = 3.140625$.)

## Single-Precision

$$\begin{aligned} pi &= 1.10010010000111111011011 \times 2^{1} \\ &= 0x1.921fb6p+1 \end{aligned}$$

This equals 3.14159274101257324218 75, which approximates pi accurately to about 8 decimal digits.

## Double-Precision

```
pi = 1.1001001000011111101101010100010001000010110100011 x 2^1
   = 0x1.921fb54442d18p+1
```

This equals 3.14159265358979311599796346854418516159057617 1875, which approximates pi accurately to about 16 decimal digits.

## Extended-Precision

```
pi = 1.1001001000011111101101010100010001000010110100011000010001101
     01 x 2^1
   = 0x1.921fb54442d1846ap+1
```

This equals

3.14159265358979323851280895940618620443274267017841339111328125

which approximates pi accurately to about 20 decimal digits.

## Quadruple-Precision

```
pi = 1.1001001000011111101101010100010001000010110100011000010001101
     10011000100110001100110001010001011100000000110111 x 2^1
   = 0x1.921fb54442d18469898cc51701b8p+1
```

This equals

3.14159265358979323846264338327950279747906809813729557300450433187429671866297553606273140758275985717773437 5

which approximates pi accurately to about 35 decimal digits.

# Euler's Number (e)

Here are the first 50 decimal digits of *e*:

2.71828182845904523536028747135266249775724709369995...

Here are the first 128 bits of *e*:

```
10.10110111111000010101000101100010100010101110110100101010011010101011111101
01110001010110001000000010011100111101001111001111000 1...
```

Here are the 128 bits again, with the rounding bits for each level of precision highlighted:

```
10.10110111111000010101000101100010100010101110110100101010011010101011111101
01110001010110001000000010011100111101001111001111000 1...
```

Here are the correctly rounded values of *e* in each of the five levels of precision:

## Half-Precision

$$e = 1.010111 \text{ x } 2^{1}$$
$$= 0x1.5cp+1$$

This equals 2.71875, which approximates *e* accurately to about 4 decimal digits.

## Single-Precision

```
e = 1.01011011111000010101 x 2^1
  = 0x1.5bf0a8p+1
```

This equals 2.718281745910644531 25, which approximates *e* accurately to about 8 decimal digits.

## Double-Precision

```
e = 1.0101101111110000101010001011000101000101011101101001 x 2^1
  = 0x1.5bf0a8b145769p+1
```

This equals 2.718281828459045090795598298427648842334747314453125, which approximates *e* accurately to about 16 decimal digits.

## Extended-Precision

```
e = 1.0101101111110000101010001011000101000101011101101001010010 11
    011 x 2^1
  = 0x1.5bf0a8b145769536p+1
```

This equals

2.718281828459045235428168107993940338928950950503349304199218 75

which approximates *e* accurately to about 20 decimal digits.

## Quadruple-Precision

```
e =
1.0101101111110000101010001011000101000101011101101001010100110101010111111
011100010101100010000000010011100111101 x 2^1
  = 0x1.5bf0a8b1457695355fb8ac404e7ap+1
```

This equals

2.718281828459045235360287471352662314358421867193548862669230860327667168019
338816975505324080586433410644531 25

which approximates *e* accurately to about 34 decimal digits.

## Addendum (July 2016)

I wanted to add a new perspective following the publishing of my articles "Number of Digits Required For Round-Trip Conversions" and "Decimal Precision of Binary Floating-Point Numbers".

For starters, I was inconsistent in how I evaluated decimal precision for the ten conversions. Using the definition that decimal precision is the maximum number of matching digits after rounding, here are the new results (changes in bold):

**Precision of pi and e By Floating-Point Format**

| Format | pi (Digits) | e (Digits) |
|---|---|---|
| Half-precision | **3** | **3** |
| Single-precision | 8 | **7** |
| Double-precision | 16 | 16 |
| Extended-precision | 20 | 20 |
| Quadruple-precision | **34** | 34 |

The four that changed all have one less digit of precision than I had stated.

Furthermore, I'd now like to consider the representations of pi and *e* in the context of the precision of their formats. For each binary precision format you can compute a range of equivalent decimal precisions:

**Range of Precision By Floating-Point Format**

| Format | Decimal Precision (Digits) |
|---|---|
| Half-precision | 3-4 |
| Single-precision | 6-8 |
| Double-precision | 15-16 |

| Format | Decimal Precision (Digits) |
|---|---|
| Extended-precision | 18-20 |
| Quadruple-precision | 33-35 |

Those values represent the range of decimal precision over the whole format; individual segments of the range will have their own unique precision. For example, in the segment $[2^1, 2^2)$, which includes both pi and *e*, the precision is:

**Precision Over $[2^1, 2^2)$ By Floating-Point Format**

| Format | Decimal Precision (Digits) |
|---|---|
| Half-precision | 3 |
| Single-precision | 7 |
| Double-precision | 16 |
| Extended-precision | 19 |
| Quadruple-precision | 34 |

You can see that some of the conversions of pi and *e* are more accurate than their segment allows. For example, the single-precision value of pi is accurate to 8 digits, but only 7 digits of precision are provided in the segment in which it resides. I call this coincidental precision. pi's proximity to a floating-point number — along with some beneficial rounding — makes its precision look greater than that for numbers in its segment in general.

EB

Numbers in computers / Convert to binary, Convert to decimal, Convert to hexadecimal, Decimals, Floating-point

## 10 comments

1. **Anonymous**

   June 22, 2012 at 4:02 pm

   this is pretty pointless...why does it matter how to convert to different floating point precisions?

2. **Rick Regan**

   June 24, 2012 at 11:35 pm

   @Anonymous (sorry for the delay — your comment was marked as spam),

   This article was a response to regular searches on my site for "pi in binary" and "e in binary". My approach was to show how they look in binary *in a computer*, in IEEE floating-point in particular. This allowed me to give examples of correct rounding and to show how different levels of binary precision correspond to different levels of decimal precision. And as a side effect, I've given hex constants that can be copied and used in code that requires correctly rounded values of these constants.

3. **vent**

   September 23, 2014 at 2:48 am

   how to calculate e in binary ? If we don't use its decimal digit and then, converting to binary.

4. **Rick Regan**

   September 23, 2014 at 8:18 am

   @vent,

   If I understand your question, just use a formula that calculates e. Any constants in the formula will be automatically converted to binary, and of course calculations will be done in binary.

5. **rathlo**

   April 3, 2015 at 3:51 am

   reply to Anonymous: it might be pointless to someone not having any idea what is truly behind the floating point numbers representation and what effect it has on the correct calculations. To find out one must do tests for different precisions (single, double, quad). The author made it easier to find out Pi end e down to the last bit for each precision mode rather

then for one to reinvent the hot water. I really admire him. He put it so much effort into the article. People like him make a difference.

---

6. **Acarya dasa**

June 20, 2015 at 12:33 pm

Pointless? How about it is fun to know and examine.

---

7. **Bernard Bang**

June 22, 2015 at 5:54 pm

I first learned binary/quinary numeral systems when I was bored by 4th grade schooling. So I joined the Early bird, science club and went to the library a lot to get it right. Since then I have taught many children as well older people, some could barely read but I was teaching them binary with ease and they still show me they kept learning later and used it in daily life to count instead of our base 10. I Teach mime, knife throwing and martial arts but am intrigued by numbers still and I just turned 70. I continue to search for ways to use binary etc. to ameliorate my life in ways I have not discovered but feel close to something great and fun if not user friendly. I am an explorer and like scientific forms of getting to an answer that is correct. Just want to compliment your approach to teaching. I always use my fingers to teach as there may not be objects or pen and pad to write where I have been. Thank you again, Bernie Bang, Mime.

---

8. **Anon2**

October 21, 2015 at 1:44 pm

“this is pretty pointless…why does it matter how to convert to different floating point precisions?”

Finite precision constraints leads to rounding errors. A better understanding of the floating point representation can help avoid pitfalls like comparing floating point numbers or properly ordering floating point operations. What is presented here is a nice reference for defining constants with the correct precision for the primitive type container.

---

9. **Pano**

   January 22, 2016 at 3:44 am

   Nothing is useless; people just want others to do the work for them and when it's not what they expected they have a tantrum.

   Like a full page of: 11.1,100,1,101,1001 etc, is what I was hoping for so I could see the distinction in each decimal digit, but I'm sure someone else would have been equally displeased with that lol. No worries. Take care, bro and don't sweat the small stuff.

---

10. **Anthony wolf**

    June 18, 2017 at 11:32 pm

    I believe combining einsteins theory, laws of thermodynamics, binary code will lead us to a true form of A.I. I do not think we can create true A.I. Because no matter how random we try to make any computational system at it's very basic level it still must be programmed by us humans. True inspiration I do not believe is still understood or able to be mathematically represented yet.

---

**Comments are closed.**