Exploring Binary

Base Conversion in PHP Using BCMath

By Rick Regan April 11th, 2009

PHP has a component called BCMath which does arbitrary-precision, decimal arithmetic. I used BCMath in my decimal/binary converter because:

- Arbitrary-precision lets it operate on very large and very small numbers, numbers that can't be represented in standard computer word sizes.
- Decimal arithmetic lets it use the same algorithms I'd use to convert between decimal and binary by hand.

(If you've written a conversion routine in standard code, especially one to convert decimal fractions to binary, you'll see the advantage of the second point.)

This article describes the implementation of my conversion routines with BCMath.

Base-Independent Terminology for Fractional Values

To work with numbers in different bases, we need terminology that is base-independent. Standard terminology is base 10-centric. Decimal numbers greater than 0 and less than 1 are called *decimal fractions* or *decimals*. The '.' part of a decimal is called a *decimal point*. Each numeral in the decimal is called a *digit*.

What do you call these entities in other bases, like binary? **Digit** is an acceptable base-independent term, although we often use the term *bit* when talking specifically about binary digits. **Radix point** is a standard base-independent term for decimal point, even if it may not be mainstream.

Unfortunately, there's no standard base-independent term for the fractional value as a whole. In the context of binary, I've seen it called "binary decimal," which is a horrible term, and "binary fraction," which is OK — things like 101/1010 notwithstanding. However, a generic term for all bases would be best. I use the term **fractional**, so I will use it in this article. It can be used generically, or modified for specific bases, like **decimal fractional** or **binary fractional**.

Four Conversion Routines

I wrote four conversion routines for the decimal/binary converter, treating integers and fractionals separately because their algorithms are different:

- **dec2bin_i()**: Convert an integer from decimal to binary.
- **bin2dec_i()**: Convert an integer from binary to decimal.
- **dec2bin_f()**: Convert a fractional from decimal to binary.
- **bin2dec_f()**: Convert a fractional from binary to decimal.

These will be covered in the sections that follow.

Note: The routines do not validate their input. All inputs are assumed to be positive numbers, with no `+' sign, commas, or spaces, and not in scientific notation. Valid examples are 367, 3.14159, and 0.1.

dec2bin_i()

- **Algorithm**
  To convert a decimal integer to binary, repeatedly divide it by 2 — using integer division — until it becomes 0. The remainders at each step, which are 0s and 1s, represent the binary number, when strung together in the reverse order in which they were generated.

  For example, 13_{10} converts to binary as follows:

  - 13/2 = 6 remainder **1**
  - 6/2 = 3 remainder **0**
  - 3/2 = 1 remainder **1**
  - 1/2 = 0 remainder **1**

  The result is 1101_2.

- **Code**

```
<?php
function dec2bin_i($decimal_i)
{
 bcscale(0);
```

```
 $binary_i = '';
 do
  {
   $binary_i = bcmod($decimal_i,'2') . $binary_i;
   $decimal_i = bcdiv($decimal_i,'2');
  } while (bccomp($decimal_i,'0'));

 return($binary_i);
}
?>
```

- Notes:
  - The BCMath routines begin with the letters 'bc'. ***They work with decimal numbers in character string form.*** Internally, BCMath converts numbers into binary-coded decimal (BCD), with each BCD digit being 8-bits instead of the usual 4 (this makes translation to and from character strings simpler).
  - bcscale sets the number of fractional digits to keep in calculations, which in this case is 0. This makes bcdiv perform integer division.
  - The line $binary_i = bcmod($decimal_i,'2') . $binary_i; prepends the current bit to the binary answer, avoiding having to reverse the string at the end.
  - bccomp is the BCMath comparison function, which in this case is used to test if $decimal_i is greater than 0.

- **Testcase**

```
<?php
echo dec2bin_i('36893488147419103232'); //2^65
?>
```

Output: A 1 followed by 65 zeros:

10000000000000000000000000000000000000000000000000000000000000000

bin2dec_i()

- **Algorithm**
  To convert a binary integer to decimal, expand it in terms of nonnegative powers of two, expressed in decimal. For example:

$$1101_2 = (1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0)_{10} .$$

You can evaluate this expression efficiently using a technique known as Horner's method. It avoids having to compute each power of two separately. Using Horner's method, you'd rewrite the expression and evaluate is as follows:

$$((1 * 2 + 1) * 2 + 0) * 2 + 1 = 13_{10}.$$

It reduces the evaluation process to two simple steps: multiply by 2, then add the next digit.

- **Code**

```
<?php
function bin2dec_i($binary_i)
{
 bcscale(0);

 $decimal_i = '0';
 for ($i = 0; $i < strlen($binary_i); $i++)
   {
    $decimal_i = bcmul($decimal_i,'2');
    $decimal_i = bcadd($decimal_i,$binary_i[$i]);
   }

 return($decimal_i);
}
?>
```

  - Notes:
    - This code implements Horner's method, except that it starts out multiplying the initial sum of 0 by 2, avoiding making a special case of the first bit.
    - bcscale sets the number of fractional digits to keep in calculations, which in this case is 0 because we are dealing only with integers.

- **Testcase**

```
<?php
echo bin2dec_i('11111111111111111111111111111111'); //2^36-1
?>
```

Output: 68719476735.

dec2bin_f()

- **Algorithm**

To convert a decimal fractional to binary, repeatedly multiply it by 2 and strip off the integer part, stopping when the fractional part becomes 0 or the desired number of fractional digits have been generated. The integer parts of the product at each step, which are 0s and 1s, represent the binary number, when strung together in the order in which they were generated.

For example, 0.8125_{10} converts to binary as follows:

- 0.8125 * 2 = **1**.625
- 0.625 * 2 = **1**.25
- 0.25 * 2 = **0**.5
- 0.5 * 2 = **1**.0

The result is 0.1101_2.

In that example, the algorithm terminated because the remaining fractional part became 0. This only happens with dyadic fractions. If you were to run the algorithm on a non-dyadic decimal fraction, like 0.1, it would generate digits forever; hence the need to quit after generating a fixed number of digits.

- **Code**

```
<?php
function dec2bin_f($decimal_f,$scale)
{
 $digitCount = 0;

 $scale = max($scale, strlen($decimal_f) - strlen('0.'));
 bcscale($scale);

 $binary_f = '0.';
 while (bccomp($decimal_f,'0') && $digitCount < $scale)
   {
    $decimal_f = bcmul($decimal_f,'2');
    if (bccomp($decimal_f,'1') >= 0)
      {
       $binary_f .= '1';
       $decimal_f = bcsub($decimal_f,'1');
      }
    else
      $binary_f .= '0';

    $digitCount++;
   }

 return($binary_f);
}
?>
```

- Notes:
  - Fractionals are of the form '0.f', where f represents a string of one or more digits (all 0s is not allowed). The value strlen(`0.') is subtracted when the number of digits after the radix point need to be counted.
  - The scale is set to be the maximum of the requested scale and the length of the decimal fractional. This enables dyadic fractions to be converted to full precision, regardless of the number of digits requested.
  - The resulting binary fractional is truncated, not rounded.

- **Testcase**

```
<?php
echo dec2bin_f('0.00000000011641532182693481445312 5',33); //2^-33
?>
```

Output: A 1 preceded by 32 0s after the radix point:

0.00000000000000000000000000000001.

bin2dec_f()

- **Algorithm**
  To convert a binary fractional to decimal, expand it in terms of negative powers of two, expressed in decimal. For example:

$$0.1101_2 = (1 * 2^{-1} + 1 * 2^{-2} + 0 * 2^{-3} + 1 * 2^{-4})_{10} .$$

  You can evaluate this expression efficiently with Horner's method, as we did for binary integers. This time, rewrite the terms ***from right to left***, then nest and evaluate as follows:

$$(((1 * 2^{-1} + 0) * 2^{-1} + 1) * 2^{-1} + 1) * 2^{-1} = 0.8125_{10} .$$

  It reduces the evaluation process to two simple steps: add the next digit, then divide by 2.

- **Code**

```
<?php
function bin2dec_f($binary_f)
```

```
{
 $scale = strlen($binary_f) - strlen('0.');
 bcscale($scale);

 $decimal_f = '0';
 for ($i = strlen($binary_f) - 1; $i >= strlen('0.'); $i--)
   {
    $decimal_f = bcadd($decimal_f,$binary_f[$i]);
    $decimal_f = bcdiv($decimal_f,'2');
   }

 return($decimal_f);
}
?>
```

- Notes:
  - This code implements Horner's method, except that it starts out adding the first bit to an initial sum of 0, avoiding making a special case of the first bit.
  - The 'for' loop works from right to left in the input binary string.
  - bcscale sets the number of fractional digits to match the length of the binary fractional (a binary fractional always converts to a decimal fractional of the same length).
  - The variable $decimal_f temporarily becomes greater than 1, whenever bcadd adds a 1 digit.

- **Testcase**

```
<?php
echo bin2dec_f('0.00000000000000000000000000000001'); //2^-33
?>
```

Output: 0.000000000116415321826934814453125.

Generalizing to Other Bases

The code above can be generalized very easily to convert between decimal and any base 2 through 9. Just change the constant '2' to the desired base and rename the variables named 'binary'. With just a little extra code, you can convert to bases greater than 10. You'd have to convert digits named with letters to their corresponding decimal values.

EB

Related

- Bicimals
- Correct Decimal To Floating-Point Using Big Integers
- Base Conversion In PHP Using Built-In Functions

Numbers in computers / Code, Convert to binary, Convert to decimal, PHP

8 comments

1. **Patrick McLaughlin**

   September 7, 2012 at 9:32 pm

   This site of yours is absolutely awesome!!!

Do you have a way I can take a total series of binary numbers from 0 through 16363 in binary and have a list of the summed decimal equivalents in a list without doing each one by hand ??

---

2. **Patrick McLaughlin**

September 7, 2012 at 9:33 pm

Revised with correct number for 1111111111111111 = 16535:

Do you have a way I can take a total series of binary numbers from 0 through 16535 in binary and have a list of the summed decimal equivalents in a list without doing each one by hand ??

---

3. **Rick Regan**

September 7, 2012 at 11:36 pm

Patrick,

Thanks. I think you mean 65535, if you're looking for 16 bits of 1s. In any case, I don't understand your question. Sorry.

---

4. **Arthur Dent**

March 13, 2014 at 3:46 pm

Rick, awesome site as it's helped me quite abit

I see you put the code to convert dec to bin but how are you converting the binary to 2's complement?

---

5. **Rick Regan**

March 13, 2014 at 11:39 pm

@Arthur,

Thanks for the feedback.

Because dec2bin_i() assumes all inputs are positive, and its output is an arbitrarily long binary string, two's complement does not come into play. Even if I allowed negative numbers, I would just prefix the output with a minus sign, rather than put it in two's complement form. I am really working with 'pure' binary numbers.

---

6. **Junichiro**

   June 3, 2018 at 11:19 pm

   this code is work for dec2bin_i with number length below 19 if the length is 20 the result is 0.

---

7. **Rick Regan**

   June 4, 2018 at 7:05 am

   @Junichiro,

   It works OK for me. Please give me an example.

---

8. **Junichiro**

   June 4, 2018 at 10:34 pm

   @Rick Regan

   My bad, I tested using int. After adding "" to my value it works already.
   sorry my bad.

   thanks and it really work

Comments are closed.