I noticed this when creating the patch at https://github.com/ruby/ruby/pull/332. I believe it affects everything from ruby 1.8 up.

BigMath::exp is implemented for negative numbers according to the identity \( E^{-x} = 1 / (E^x) \). At bigdecimal.c:2742 (in GitHub commit 258acf3, in the implementation of BigMath_s_exp), the code flips the sign on the input argument for negative numbers. Later, it returns the reciprocal of the result.

Problem: When the first argument is a BigDecimal (or, presumably, Bignum), the original argument is modified, so its sign bit remains flipped. Hence:

```ruby
2.0.0-p195 :001 > require 'bigdecimal/math'
=> true
2.0.0-p195 :002 > x = BigDecimal(-1)
=> #
2.0.0-p195 :003 > BigMath.exp(x, 20)
=> #
2.0.0-p195 :004 > x
=> #
```

Here, \( x \) is modified when used as an argument to BigMath.exp AND the answer is wrong. I have already submitted the patch mentioned above for the latter problem, but I'm not sure what the appropriate fix would be for the sign modification. Just resetting the sign bit on exit would be easy but not thread safe; BigMath.exp really shouldn't be modifying the argument at all. But copying the whole argument is potentially wasteful if the precision is high.

I suspect that the special calculation track for negative values is actually not needed at all. Without the patch I just submitted, BigMath.exp is reliably returning the reciprocal of the correct answer, which means that it's properly calculating the correct answer by using the (negative) VALUE \( x \) passed in as the original argument -- at least for immediate values. So perhaps the basic iteration loop is just as valid for negative exponents as it is already implemented.

History

#1 - 06/19/2013 08:55 AM - GSnyder (Garth Snyder)

It appears that the iteration loop in BigMath_s_exp is just calculating the series expansion \((X ** N) / N!\) from \( N = 0 \) until the incremental term vanishes below the precision threshold. I don't see any reason this would not be equally valid for negative \( X \). The code already checks the increment \( d \) with VpIsZero(), which accommodates either negative or positive zero.

#2 - 06/19/2013 07:42 PM - naruse (Yui NARUSE)

- Category set to ext
- Status changed from Open to Assigned
- Assignee set to mrkn (Kenta Murata)
- Target version set to 2.1.0

#3 - 06/20/2013 04:34 PM - GSnyder (Garth Snyder)

In the original report, I stated that the answer was wrong; that is incorrect. It's wrong (in the current trunk) if the first argument is a negative, immediate type, but BigDecimal arguments give correct answers. They just become modified in the process.

I experimented with just removing the special handling for negative arguments in BigMath.exp(), but there is in fact something odd about the...
calculation mechanism that gives incorrect results.

#4 - 01/30/2014 06:17 AM - hsbt (Hiroshi SHIBATA)
- Target version changed from 2.1.0 to 2.2.0

#5 - 01/05/2018 09:00 PM - naruse (Yui NARUSE)
- Target version deleted (2.2.0)

#6 - 08/10/2019 05:31 AM - jeremyevans0 (Jeremy Evans)
- Backport changed from 1.9.3: UNKNOWN, 2.0.0: UNKNOWN to 2.6: REQUIRED
- File bigdecimal-exp-no-mutate-8542.patch added

This bug is still present in the master branch. This is actually a fairly significant bug since Ruby 2.6, as starting in Ruby 2.6, BigDecimal instances are frozen and should never be mutated. Attached is a patch that fixes this issue. I think this fix should be backported to 2.6 because Ruby should not allow modification of frozen objects.

#7 - 08/15/2019 07:22 PM - jeremyevans0 (Jeremy Evans)
As bigdecimal upstream is in a separate repository, I submitted a pull request for this: https://github.com/ruby/bigdecimal/pull/150.

#8 - 10/07/2019 09:25 PM - jeremyevans0 (Jeremy Evans)
- Status changed from Assigned to Closed

Pull request was merged: https://github.com/ruby/bigdecimal/pull/150

Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigdecimal-exp-no-mutate-8542.patch</td>
<td>1.63 KB</td>
<td>08/10/2019</td>
<td>jeremyevans0 (Jeremy Evans)</td>
</tr>
</tbody>
</table>