Subject: How Computers Represent Floats
Posted by davidf on Wed, 29 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Oh, dear. :-(

I have occasion to recall a discussion posted in this forum some time ago about how computers represent floating point numbers. How they appear inaccurate, etc.

I *know* I saved it, but I've searched on just about every keyword I can think of on my local machines and in Dejanews and I can't find what I am looking for. (It's possible I dreamed the whole exchange. Stranger things have happened.)

If anyone saved the discussion (or even recalls it enough to supply me with some likely keywords to search on), I would be grateful.

Cheers,

David

P.S. In my dream (apparently) someone who is not a frequent poster to this newsgroup published a fabulously informative article on the subject.

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: How Computers Represent Floats
Posted by Karl Schultz on Fri, 01 Dec 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Here's another useful table if anyone is interested in pondering the limits of floating point representations:

	32-bit(single)	64-bit(double)
Mantissa	24 bits	53 bits
Exponent	8 bits	11 bits
EPS	1.19209e-007	2.22045e-016

Min	1.17549e-038	2.22507e-308
Max	3.40282e+038	1.79769e+308
Decimal Places	6	15

	80-bit	128-bit(quad)
Mantissa	65 bits	113 bits
Exponent	15 bits	15 bits
EPS	1.08420e-019	1.92593e-034
Min	3.36210e-4932	3.36210e-4932
Max	1.18973e+4932	1.18973e+4932
Decimal Places	18	33

The mantissa bit counts include the sign bit.

I find it a little interesting that the number of bits in the exponent remains the same between 80-bit and 128-bit.

I also think that the EPS (machine epsilon or machine precision) is probably one of the most important values. It is the smallest value that you can add to 1.0 and still have the result be something other than 1.0. This can give you an idea of how closely you can resolve (differentiate) floating point values at a given magnitude.

For example, see what this does in IDL 5.3:
PLOT,FINDGEN(100),FINDGEN(100)+2d8,YSTYLE=3

IDL 5.4 gives different results because PLOT works in double precision. You can "simulate" the old 5.3 behavior with:

PLOT,FINDGEN(100),FLOAT(FINDGEN(100)+2d8), YSTYLE=3

Karl

Subject: Re: How Computers Represent Floats
Posted by colinr on Fri, 01 Dec 2000 08:00:00 GMT
View Forum Message <> Reply to Message

On Thu, 30 Nov 2000 13:23:41 -0700,
William B. Clodius <wclodius@lanl.gov> wrote:
>
>
> "William B. Clodius" wrote:
>> <snip> IEEE 754 requires that all intermediate calculations

>> be performed a higher precision so
> Ignore the above incomplete sentence. What I originally attempted to
> write was covered later.
>>
>> <snip>
>
> Some other surprises.
>
> The definition of the IEEE 754 mantisa, an integer with values from
> 2^n_mant to 2*2^n_mant-1, where n_mant is the number of bits available
> for the mantisa, is termed a normalized number. This is error prone for
> very small numbers. IEEE 754 mandates that there be available for such
> small numbers what are termed denorms where the mantissa is interpreted
> as an integer from 0 to 2^n_mant, so that accuracy degrades gradually
> for such values. However, this complicates the implementation of the
> floating point, so some processors, e.g., the DEC Alpha make this
> available only in software at a greatly reduced performance.

This sounds like it relates to my most recent problem - generating
real input files with IDL for a DEC Alpha fortran program. The
fortran program had big problems manipulating small numbers
generated by the IDL and I had to pepper the IDL code with WHERE
statements to set all very small numbers to zero. Has anyone else
seen stuff like this?

--
Colin Rosenthal
Astrophysics Institute
University of Oslo
