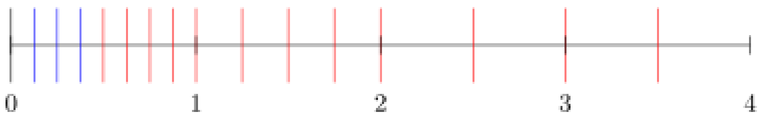
Denormal number

In computer science, **denormal numbers** or **denormalized numbers** (now often called **subnormal numbers**) fill the underflow gap around zero in floating-point arithmetic. Any non-zero number with magnitude smaller than the smallest normal number is "subnormal".



An unaugmented floating-point system would contain only normalized numbers (indicated in red). Allowing denormalized numbers (blue) extends the system's range.

In a normal floating-point value, there are no leading zeros in the significand; instead leading zeros are removed by adjusting the exponent. So 0.0123 would be written as 1.23×10^{-2}. Denormal numbers are numbers where this representation would result in an exponent that is below the smallest representable exponent (the exponent usually having a limited range). Such numbers are represented using leading zeros in the significand.

The significand (or mantissa) of an IEEE floating-point number is the part of a floating-point number that represents the significant digits. For a positive normalised number it can be represented as $m_0.m_1m_2m_3...m_{p-2}m_{p-1}$ (where m represents a significant digit, and p is the precision) with non-zero m_0. Notice that for a binary radix, the leading binary digit is always 1. In a **denormal number**, since the exponent is the least that it can be, zero is the leading significand digit ($0.m_1m_2m_3...m_{p-2}m_{p-1}$), allowing the representation of numbers closer to zero than the smallest normal number. A floating-point number may be recognized as denormal whenever its exponent is the least value possible.

By filling the underflow gap like this, significant digits are lost, but not as abruptly as when using the *flush to zero on underflow* approach (discarding all significant digits when underflow is reached). Hence the production of a denormal number is sometimes called **gradual underflow** because it allows a calculation to lose precision slowly when the result is small.

In IEEE 754-2008, denormal numbers are renamed *subnormal numbers* and are supported in both binary and decimal formats. In binary interchange formats, subnormal numbers are encoded with a biased exponent of 0, but are interpreted with the value of the smallest allowed exponent, which is one greater (i.e., as if it were encoded as a 1). In decimal interchange formats they require no special encoding because the format supports unnormalized numbers directly.

Mathematically speaking, the normalized floating-point numbers of a given sign are roughly logarithmically spaced, and as such any finite-sized normal float cannot include zero. The denormal floats are a linearly spaced set of values, which span the gap between the negative and positive normal floats.

Contents

Background

Performance issues

Disabling denormal floats at the code level
See also
References
Further reading

Background

Denormal numbers provide the guarantee that addition and subtraction of floating-point numbers never underflows; two nearby floating-point numbers always have a representable non-zero difference. Without gradual underflow, the subtraction $a - b$ can underflow and produce zero even though the values are not equal. This can, in turn, lead to division by zero errors that cannot occur when gradual underflow is used.[1]

Denormal numbers were implemented in the Intel 8087 while the IEEE 754 standard was being written. They were by far the most controversial feature in the K-C-S format proposal that was eventually adopted,[2] but this implementation demonstrated that denormals could be supported in a practical implementation. Some implementations of floating-point units do not directly support denormal numbers in hardware, but rather trap to some kind of software support. While this may be transparent to the user, it can result in calculations that produce or consume denormal numbers being much slower than similar calculations on normal numbers.

Performance issues

Some systems handle denormal values in hardware, in the same way as normal values. Others leave the handling of denormal values to system software, only handling normal values and zero in hardware. Handling denormal values in software always leads to a significant decrease in performance. When denormal values are entirely computed in hardware, implementation techniques exist to allow their processing at speeds comparable to normal numbers;[3] however, the speed of computation is significantly reduced on many modern processors; in extreme cases, instructions involving denormal operands may run as much as 100 times slower.[4][5]

This speed difference can be a security risk. Researchers showed that it provides a timing side channel that allows a malicious web site to extract page content from another site inside a web browser.[6]

Some applications need to contain code to avoid denormal numbers, either to maintain accuracy, or in order to avoid the performance penalty in some processors. For instance, in audio processing applications, denormal values usually represent a signal so quiet that it is out of the human hearing range. Because of this, a common measure to avoid denormals on processors where there would be a performance penalty is to cut the signal to zero once it reaches denormal levels or mix in an extremely quiet noise signal.[7] Other methods of preventing denormal numbers include adding a DC offset, quantizing numbers, adding a Nyquist signal, etc.[8] Since the SSE2 processor extension, Intel has provided such a functionality in CPU hardware, which rounds denormalized numbers to zero.[9]

Disabling denormal floats at the code level

Intel's C and Fortran compilers enable the denormals-are-zero (DAZ) and flush-to-zero (FTZ) flags for SSE by default for optimization levels higher than -O0.[10] The effect of DAZ is to treat denormal input arguments to floating-point operations as zero, and the effect of FTZ is to return zero instead of a denormal float for operations that would result in a denormal float, even if the input arguments are not themselves denormal. clang and gcc have varying default states depending on platform and optimization level. A non-C99-compliant method of enabling the DAZ and FTZ flags on targets supporting SSE is given below, but is not widely supported. It is known to work on Mac OS X since at least 2006.[11]

The DAZ flag can be set as following. However, to deal with performance issue introduced during computation the FTZ flag should be enabled as well.

```
#include <fenv.h>
fesetenv(FE_DFL_DISABLE_SSE_DENORMS_ENV);
```

For other SSE instruction-set platforms where the C library has not yet implemented the above flag, the following may work:[12]

```
#include <xmmintrin.h>
//DAZ
_mm_setcsr( _mm_getcsr() | 0x0040 );
//FTZ
_mm_setcsr( _mm_getcsr() | 0x8000 );
```

The _MM_SET_DENORMALS_ZERO_MODE and _MM_SET_FLUSH_ZERO_MODE macro wraps a better interface for the code above. It allows switching the mode on or off, while preserving any other configuration in the CSR.[13]

```
//To enable DAZ
#include <pmmintrin.h>
_MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
//To enable FTZ
#include <xmmintrin.h>
_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);
```

Most compilers will already provide the previous macro by default, otherwise the following code snippet can be used:

```
#define _MM_DENORMALS_ZERO_MASK   0x0040
#define _MM_DENORMALS_ZERO_ON     0x0040
#define _MM_DENORMALS_ZERO_OFF    0x0000

#define _MM_SET_DENORMALS_ZERO_MODE(mode)                                   \
            _mm_setcsr((_mm_getcsr() & ~_MM_DENORMALS_ZERO_MASK) | (mode))
#define _MM_GET_DENORMALS_ZERO_MODE()                                       \
            (_mm_getcsr() & _MM_DENORMALS_ZERO_MASK)
```

The default denormal behavior is ABI, and therefore well-behaved software should save and restore the denormal mode before returning to the caller or calling out to unsuspecting library/OS code.

See also

- Logarithmic number system

References

1. William Kahan. "IEEE 754R meeting minutes, 2002" (https://web.archive.org/web/20161015154158/http://grouper.ieee.org/groups/754/meeting-minutes/02-09-19.html#underflow). Archived from the original (http://grouper.ieee.org/groups/754/meeting-minutes/02-09-19.html#underflow) on 15 October 2016. Retrieved 29 Dec 2013.
2. "An Interview with the Old Man of Floating-Point" (http://www.eecs.berkeley.edu/~wkahan/ieee754status/754story.html). University of California, Berkeley.
3. Schwarz, E.M.; Schmookler, M.; Son Dao Trong (July 2005). "FPU Implementations with Denormalized Numbers" (http://www.acsel-lab.com/arithmetic/arith16/papers/ARITH16_Schwarz.pdf) (PDF). *IEEE Transactions on Computers*. **54** (7): 825–836. doi:10.1109/TC.2005.118 (https://doi.org/10.1109%2FTC.2005.118).
4. Dooley, Isaac; Kale, Laxmikant (2006-09-12). "Quantifying the Interference Caused by Subnormal Floating-Point Values" (http://charm.cs.uiuc.edu/papers/SubnormalOSIHPA06.pdf) (PDF). Retrieved 2010-11-30.
5. Fog, Agner. "Instruction tables: Lists of instruction latencies, throughputs and microoperation breakdowns for Intel, AMD and VIA CPUs" (http://www.agner.org/optimize/instruction_tables.pdf) (PDF). Retrieved 2011-01-25.
6. Andrysco, Marc; Kohlbrenner, David; Mowery, Keaton; Jhala, Ranjit; Lerner, Sorin; Shacham, Hovav. "On Subnormal Floating Point and Abnormal Timing" (https://cseweb.ucsd.edu/~dkohlbre/papers/subnormal.pdf) (PDF). Retrieved 2015-10-05.
7. Serris, John (2002-04-16). "Pentium 4 denormalization: CPU spikes in audio applications" (https://web.archive.org/web/20120225091101/http://phonophunk.com/articles/pentium4-denormalization.php). Archived from the original (http://phonophunk.com/articles/pentium4-denormalization.php) on February 25, 2012. Retrieved 2015-04-29.
8. de Soras, Laurent (2005-04-19). "Denormal numbers in floating point signal processing applications" (http://ldesoras.free.fr/doc/articles/denormal-en.pdf) (PDF).
9. Casey, Shawn (2008-10-16). "x87 and SSE Floating Point Assists in IA-32: Flush-To-Zero (FTZ) and Denormals-Are-Zero (DAZ)" (http://software.intel.com/en-us/articles/x87-and-sse-floating-point-assists-in-ia-32-flush-to-zero-ftz-and-denormals-are-zero-daz/). Retrieved 2010-09-03.
10. "Intel® MPI Library – Documentation" (http://software.intel.com/sites/products/documentation/hpc/composerxe/en-us/2011Update/fortran/win/fpops/common/fpops_reduce_denorm.htm). Intel.
11. "Re: Macbook pro performance issue" (https://web.archive.org/web/20160826010613/https://lists.apple.com/archives/perfoptimization-dev/2006/May/msg00013.html). Apple Inc. Archived from the original (https://lists.apple.com/archives/perfoptimization-dev/2006/May/msg00013.html) on 2016-08-26.
12. "Re: Changing floating point state (Was: double vs float performance)" (https://web.archive.org/web/20140115124313/http://lists.apple.com/archives/perfoptimization-dev/2007/Jun/msg00025.html). Apple Inc. Archived from the original (http://lists.apple.com/archives/perfoptimization-dev/2007/Jun/msg00025.html) on 2014-01-15. Retrieved 2013-01-24.
13. "C++ Compiler for Linux* Systems User's Guide" (https://software.intel.com/sites/default/files/ae/4f/6320). Intel.

Further reading

- Eric Schwarz, Martin Schmookler and Son Dao Trong (June 2003). "Hardware Implementations of Denormalized Numbers" (http://www.ece.ucdavis.edu/acsel/arithmetic/arith16/papers/ARITH16_Schwarz.pdf) (PDF). *Proceedings 16th IEEE Symposium on Computer Arithmetic (Arith16)*. 16th IEEE Symposium on Computer Arithmetic (http://www.dec.usc.es/arith16/). IEEE Computer Society. pp. 104–111. ISBN 0-7695-1894-X.

See also various papers on William Kahan's web site [1] (http://www.cs.berkeley.edu/~wkahan/) for examples of where denormal numbers help improve the results of calculations.