WIKIPEDIA

Half-precision floating-point format

In computing, **half precision** is a binary floating-point computer number format that occupies 16 bits (two bytes in modern computers) in computer memory.

In the IEEE 754-2008 standard, the 16-bit base-2 format is referred to as **binary16**. It is intended for storage of floating-point values in applications where higher precision is not essential for performing arithmetic computations.

Although implementations of the IEEE Half-precision floating point are relatively new, several earlier 16-bit floating point formats have existed including that of Hitachi's HD61810 DSP[1] of 1982, Scott's WIF[2] and the 3dfx Voodoo Graphics processor.[3]

Nvidia and Microsoft defined the **half** datatype in the Cg language, released in early 2002, and implemented it in silicon in the GeForce FX, released in late 2002.[4] ILM was searching for an image format that could handle a wide dynamic range, but without the hard drive and memory cost of floating-point representations that are commonly used for floating-point computation (single and double precision).[5] The hardware-accelerated programmable shading group led by John Airey at SGI (Silicon Graphics) invented the s10e5 data type in 1997 as part of the 'bali' design effort. This is described in a SIGGRAPH 2000 paper[6] (see section 4.3) and further documented in US patent 7518615.[7]

This format is used in several computer graphics environments including MATLAB, OpenEXR, JPEG XR, GIMP, OpenGL, Cg, Direct3D, and D3DX. The advantage over 8-bit or 16-bit binary integers is that the increased dynamic range allows for more detail to be preserved in highlights and shadows for images. The advantage over 32-bit single-precision binary formats is that it requires half the storage and bandwidth (at the expense of precision and range).[5]

The F16C extension allows x86 processors to convert half-precision floats to and from single-precision floats.

Depending on the computer half-precision can be over an order of magnitude faster than double precision, e.g. 37 PFLOPS vs. for half 550 "AI-PFLOPS (Half Precision)".[8]

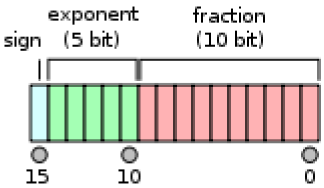
Contents

IEEE 754 half-precision binary floating-point format: binary16

The IEEE 754 standard specifies a **binary16** as having the following format:

- Sign bit: 1 bit
- Exponent width: 5 bits
- Significand precision: 11 bits (10 explicitly stored)

The format is laid out as follows:



The format is assumed to have an implicit lead bit with value 1 unless the exponent field is stored with all zeros. Thus only 10 bits of the significand appear in the memory format but the total precision is 11 bits. In IEEE 754 parlance, there are 10 bits of significand, but there are 11 bits of significand precision ($\log_{10}(2^{11}) \approx 3.311$ decimal digits, or 4 digits ± slightly less than 5 units in the last place).

Exponent encoding

The half-precision binary floating-point exponent is encoded using an offset-binary representation, with the zero offset being 15; also known as exponent bias in the IEEE 754 standard.

- $E_{min} = 00001_2 - 01111_2 = -14$
- $E_{max} = 11110_2 - 01111_2 = 15$
- Exponent bias = $01111_2 = 15$

Thus, as defined by the offset binary representation, in order to get the true exponent the offset of 15 has to be subtracted from the stored exponent.

The stored exponents 00000_2 and 11111_2 are interpreted specially.

Exponent	Significand = zero	Significand ≠ zero	Equation
00000_2	zero, −0	subnormal numbers	$(-1)^{\text{signbit}} \times 2^{-14} \times 0.\text{significantbits}_2$
00001_2, ..., 11110_2	normalized value		$(-1)^{\text{signbit}} \times 2^{\text{exponent}-15} \times 1.\text{significantbits}_2$
11111_2	±infinity	NaN (quiet, signalling)	

The minimum strictly positive (subnormal) value is $2^{-24} \approx 5.96 \times 10^{-8}$. The minimum positive normal value is $2^{-14} \approx 6.10 \times 10^{-5}$. The maximum representable value is $(2-2^{-10}) \times 2^{15} = 65504$.

Half precision examples

These examples are given in bit representation of the floating-point value. This includes the sign bit, (biased) exponent, and significand.

$\texttt{0 00000 0000000001}_2 = \texttt{0001}_{16} = 2^{-14} \times (0 + \frac{1}{1024}) \approx 0.000000059605$
(smallest positive subnormal number)

$\texttt{0 00000 1111111111}_2 = \texttt{03ff}_{16} = 2^{-14} \times (0 + \frac{1023}{1024}) \approx 0.000060976$
(largest subnormal number)

$\texttt{0 00001 0000000000}_2 = \texttt{0400}_{16} = 2^{-14} \times (1 + \frac{0}{1024}) \approx 0.000061035$
(smallest positive normal number)

$\texttt{0 11110 1111111111}_2 = \texttt{7bff}_{16} = 2^{15} \times (1 + \frac{1023}{1024}) = 65504$
(largest normal number)

$\texttt{0 01110 1111111111}_2 = \texttt{3bff}_{16} = 2^{-1} \times (1 + \frac{1023}{1024}) \approx 0.99951$
(largest number less than one)

$\texttt{0 01111 0000000000}_2 = \texttt{3c00}_{16} = 2^{0} \times (1 + \frac{0}{1024}) = 1$
(one)

$\texttt{0 01111 0000000001}_2 = \texttt{3c01}_{16} = 2^{0} \times (1 + \frac{1}{1024}) \approx 1.001$
(smallest number larger than one)

$\texttt{0 01101 0101010101}_2 = \texttt{3555}_{16} = 2^{-2} \times (1 + \frac{341}{1024}) = 0.333251953125$
(equal to 1/3)

$\texttt{1 10000 0000000000}_2 = \texttt{c000}_{16} = -2$

$\texttt{0 00000 0000000000}_2 = \texttt{0000}_{16} = 0$
$\texttt{1 00000 0000000000}_2 = \texttt{8000}_{16} = -0$

$\texttt{0 11111 0000000000}_2 = \texttt{7c00}_{16} = \text{infinity}$
$\texttt{1 11111 0000000000}_2 = \texttt{fc00}_{16} = -\text{infinity}$

By default, 1/3 rounds down like for double precision, because of the odd number of bits in the significand. So the bits beyond the rounding point are `0101...` which is less than 1/2 of a unit in the last place.

Precision limitations on decimal values in [0, 1]

- Decimals between 2^{-24} (minimum positive subnormal) and 2^{-14} (maximum subnormal): fixed interval 2^{-24}
- Decimals between 2^{-14} (minimum positive normal) and 2^{-13}: fixed interval 2^{-24}

- Decimals between 2^{-13} and 2^{-12}: fixed interval 2^{-23}
- Decimals between 2^{-12} and 2^{-11}: fixed interval 2^{-22}
- Decimals between 2^{-11} and 2^{-10}: fixed interval 2^{-21}
- Decimals between 2^{-10} and 2^{-9}: fixed interval 2^{-20}
- Decimals between 2^{-9} and 2^{-8}: fixed interval 2^{-19}
- Decimals between 2^{-8} and 2^{-7}: fixed interval 2^{-18}
- Decimals between 2^{-7} and 2^{-6}: fixed interval 2^{-17}
- Decimals between 2^{-6} and 2^{-5}: fixed interval 2^{-16}
- Decimals between 2^{-5} and 2^{-4}: fixed interval 2^{-15}
- Decimals between 2^{-4} and 2^{-3}: fixed interval 2^{-14}
- Decimals between 2^{-3} and 2^{-2}: fixed interval 2^{-13}
- Decimals between 2^{-2} and 2^{-1}: fixed interval 2^{-12}
- Decimals between 2^{-1} and 2^{-0}: fixed interval 2^{-11}

Precision limitations on decimal values in [1, 2048]

- Decimals between 1 and 2: fixed interval 2^{-10} ($1+2^{-10}$ is the next largest float after 1)
- Decimals between 2 and 4: fixed interval 2^{-9}
- Decimals between 4 and 8: fixed interval 2^{-8}
- Decimals between 8 and 16: fixed interval 2^{-7}
- Decimals between 16 and 32: fixed interval 2^{-6}
- Decimals between 32 and 64: fixed interval 2^{-5}
- Decimals between 64 and 128: fixed interval 2^{-4}
- Decimals between 128 and 256: fixed interval 2^{-3}
- Decimals between 256 and 512: fixed interval 2^{-2}
- Decimals between 512 and 1024: fixed interval 2^{-1}
- Decimals between 1024 and 2048: fixed interval 2^{0}

Precision limitations on integer values

- Integers between 0 and 2048 can be exactly represented (and also between −2048 and 0)
- Integers between 2048 and 4096 round to a multiple of 2 (even number)
- Integers between 4096 and 8192 round to a multiple of 4
- Integers between 8192 and 16384 round to a multiple of 8
- Integers between 16384 and 32768 round to a multiple of 16
- Integers between 32768 and 65519 round to a multiple of 32[9]
- Integers above 65519 are rounded to "infinity" if using round-to-even, or above 65535 if using round-to-zero, or above 65504 if using round-to-infinity.

ARM alternative half-precision

ARM processors support (via a floating point control register bit) an "alternative half-precision" format, which does away with the special case for an exponent value of 31 (11111_2).[10] It is almost identical to the IEEE format, but there is no encoding for infinity or NaNs; instead, an exponent of 31 encodes normalized numbers in the range 65536 to 131008.

See also

- bfloat16 floating-point format: Alternative 16-bit floating-point format with 8 bits of exponent and 7 bits of mantissa
- IEEE 754: IEEE standard for floating-point arithmetic (IEEE 754)
- ISO/IEC 10967, Language Independent Arithmetic
- Primitive data type
- RGBE image format

References

1. "hitachi :: dataBooks :: HD61810 Digital Signal Processor Users Manual" (https://archive.org/details/bitsavers_hitachidatlSignalProcessorUsersManual_4735688). *Archive.org*. Retrieved 2017-07-14.
2. Scott, Thomas J. (March 1991). "Mathematics and Computer Science at Odds over Real Numbers" (https://dl.acm.org/citation.cfm?id=107029). *SIGCSE '91 Proceedings of the twenty-second SIGCSE technical symposium on Computer science education*. **23** (1): 130–139.
3. "/home/usr/bk/glide/docs2.3.1/GLIDEPGM.DOC" (http://www.gamers.org/dEngine/xf3D/glide/glidepgm.htm). *Gamers.org*. Retrieved 2017-07-14.
4. "vs_2_sw" (https://developer.download.nvidia.com/cg/vs_2_sw.html). *Cg 3.1 Toolkit Documentation*. Nvidia. Retrieved 17 August 2016.
5. "OpenEXR" (http://www.openexr.com/about.html). OpenEXR. Retrieved 2017-07-14.
6. Mark S. Peercy; Marc Olano; John Airey; P. Jeffrey Ungar. "Interactive Multi-Pass Programmable Shading" (https://people.csail.mit.edu/ericchan/bib/pdf/p425-peercy.pdf) (PDF). *People.csail.mit.edu*. Retrieved 2017-07-14.
7. "Patent US7518615 - Display system having floating point rasterization and floating point ... - Google Patents" (https://www.google.com/patents/US7518615). *Google.com*. Retrieved 2017-07-14.
8. "About ABCI - About ABCI | ABCI" (https://abci.ai/en/about_abci/). *abci.ai*. Retrieved 2019-10-06.
9. "Mediump float calculator" (https://oletus.github.io/float16-simulator.js/). Retrieved 2016-07-26. Half precision floating point calculator
10. "Half-precision floating-point number support" (http://infocenter.arm.com/help/topic/com.arm.doc.dui0205j/CIHGAECI.html). *RealView Compilation Tools Compiler User Guide*. 10 December 2010. Retrieved 2015-05-05.

Further reading

- Khronos Vulkan signed 16-bit floating point format (https://www.khronos.org/registry/DataFormat/specs/1.2/dataformat.1.2.html#16bitfp)

External links

- Minifloats (https://www.mrob.com/pub/math/floatformats.html#minifloat) (in *Survey of Floating-Point Formats*)
- OpenEXR site (http://www.openexr.org/)
- Half precision constants (https://technet.microsoft.com/en-us/library/bb147247(v=vs.85).aspx) from D3DX
- OpenGL treatment of half precision (https://web.archive.org/web/20170531074746/http://oss.sgi.com/projects/ogl-sample/registry/ARB/half_float_pixel.txt)
- Fast Half Float Conversions (http://www.fox-toolkit.org/ftp/fasthalffloatconversion.pdf)
- Analog Devices variant (http://www.analog.com/static/imported-files/processor_manuals/ADSP_2136x_PGR_rev1-1.pdf) (four-bit exponent)
- C source code to convert between IEEE double, single, and half precision can be found here (https://www.mathworks.com/matlabcentral/fileexchange/23173)
- Java source code for half-precision floating-point conversion (https://stackoverflow.com/a/6162687/237321)
- Half precision floating point for one of the extended GCC features (https://gcc.gnu.org/onlinedocs/gcc/Half-Precision.html)