Topic 23

Floating-Point Representation

Normalization

Base 10:

$$26.73_{10} = 2 \times 10^{1} + 6 \times 10^{0} + 7 \times 10^{-1} + 3 \times 10^{-2}$$

$$26.73_{10} = 20 + 6 + \frac{7}{10} + \frac{3}{100}$$

Normalized:

$$2.673_{10} \times 10^{1}$$

Base 2:

$$8.25_{10}$$

Normalized:

$$1000.01_{2} = 1.00001_{2} \times 2^{3}$$

Floating Point

Representation for non-integer numbers

Including very small and very large numbers

Like scientific notation

-2.34×10^{56} — Normalized

$+0.002 \times 10^{-4}$

$+987.02 \times 10^{9}$ — Not normalized

In binary

$\pm 1.xxxxxxx_2 \times 2^{yyyy}$

Types float and double in C

Floating Point Standard

Defined by IEEE Std 754-1985

Developed in response to divergence of representations

- Portability issues for scientific code

Now almost universally adopted

Two representations

- Single precision (32-bit)
- Double precision (64-bit)

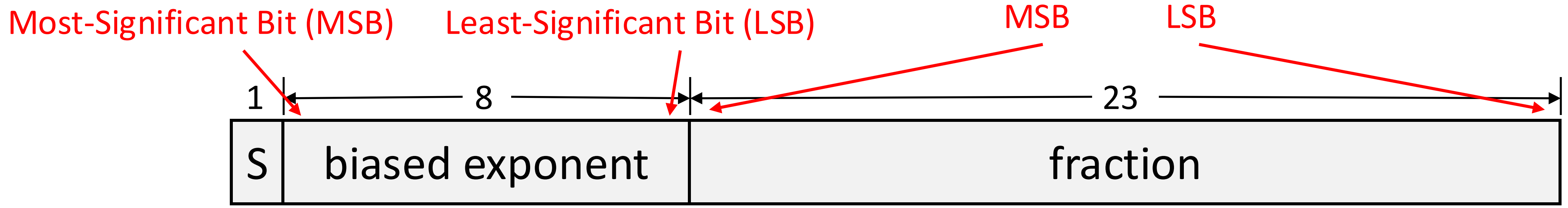
Floating Point Representation

The IEEE Std 754-1985 32-bit floating point representation uses:

1 bit for the sign (positive or negative)

8 bits for the range (exponent field)

23 bits for the precision (fraction field)



$N = \{($

Floating Point Representation (example)

1	8	23
S	biased exponent	fraction

$$N = \begin{cases} (-1)^S \times 1.fraction \times 2^{biasedexponent-127}, & 1 \le biasedexponent \le 254 \\ (-1)^S \times 0.fraction \times 2^{-126}, & biasedexponent = 0 \end{cases}$$

How is the number $-6\frac{5}{8}$ represented in floating point?

$$-6\frac{5}{8} = -\left(4 + 2 + \frac{4}{8} + \frac{1}{8}\right) = -\left(4 + 2 + \frac{1}{2} + \frac{1}{8}\right)$$

$$-(1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3})$$

$$-(110.101_2) = -(1.10101_2 \times 2^2)$$

The biasedexponent is given by: biasedexponent = actualexponent + 127

biasedexponent = 2 + 127 = 129

1 10000001 10101000000000000000000

Floating Point Representation (example)

1	8	23
S	biasedexponent	fraction

$$N = \begin{cases} (-1)^S \times 1.fraction \times 2^{biasedexponent-127}, & 1 \le biasedexponent \le 254 \\ (-1)^S \times 0.fraction \times 2^{-126}, & biasedexponent = 0 \end{cases}$$

What is the decimal value of the following floating-point number?

$$0\underbrace{011\ 1101\ 1}_{\text{biasedexponent}}000\ 0000\ 0000\ 0000\ 0000\ 0000$$

$$biasedexponent = 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 = 64 + 32 + 16 + 8 + 2 + 1 = 123$$

$$biasedexponent = 2^7 - 2^3 + 2^1 + 2^0 = 128 - 8 + 3 = 123$$

$$actualexponent = biasedexponent - 127 = 123 - 127 = -4$$

$$N = (-1)^0 \times 1.0_2 \times 2^{123-127} = 1.0 \times 2^{-4} = \frac{1}{16}$$

Answering Dylan Ashley's Question (Fall 2014)

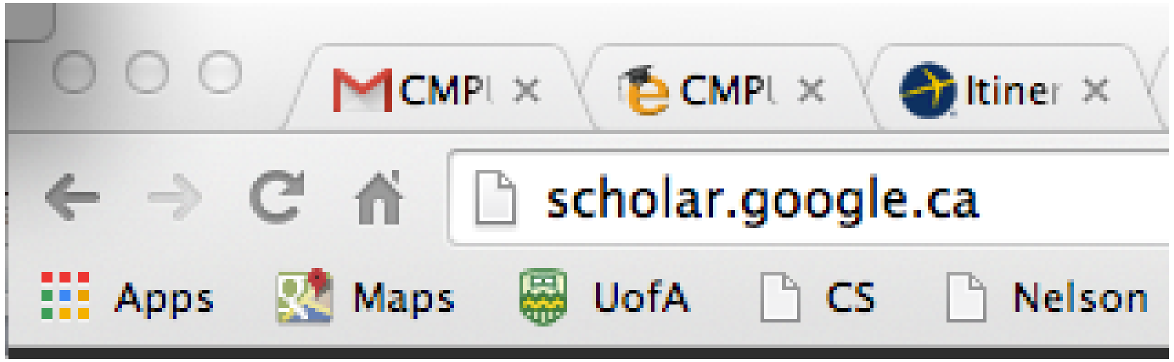
Dylan Ashley (He/Him) · 2nd

PhD student studying reinforcement learning with Jürgen Schmidhuber. MSc with Richard Sutton at the University of Alberta. Sometimes amateur photographer.

Lugano, Ticino, Switzerland · Contact info

Why does the IEEE standard use the bias format for the exponent instead of a two's-complement representation?

How to Find the Answer



Google floating-point representation

Scholar About 189,000 results (**0.04** sec)

Articles
Case law
My library

Any time
Since 2014
Since 2013
Since 2010
Custom range...

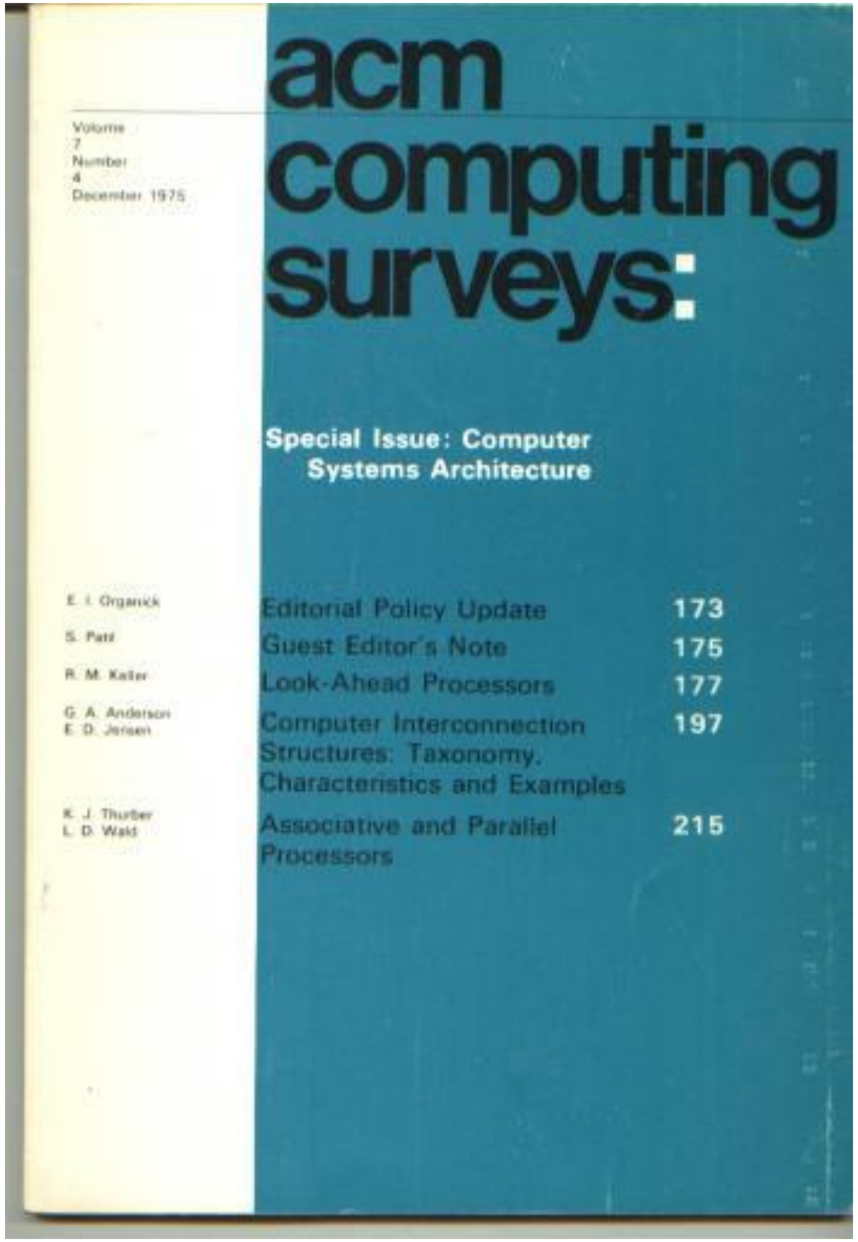
Sort by relevance
Sort by date

☑ include patents
☑ include citations

[PDF] An experimental comparison of binary and **floating point representations** in g
algorithms.
CZ Janikow, Z Michalewicz - ICGA, 1991 - cs.umsl.edu
Abstract Genetic Algorithms (GAs) are innovative search algorithms based on natural phenomena, whose main advantages lie in great robustness and problem independence. So far, GAs were most successful in parameter optimization domains; however, even there **...**
Cited by 515 Related articles All 3 versions Cite Save

What every computer scientist should know about **floating-point** arithmetic
D Goldberg - ACM Computing Surveys (CSUR), 1991 - dl.acm.org
... Numbers that are out of range will, however, be discussed in Sec- tions 2.2.2 and 2.2.4. **Floating-point representations** are not necessarily unique. **...** (1)], the **representation** is said to be normalized. The **floating-point** num- ber 1.00 x 10-1 is normalized, whereas 0.01 x 101 is not. **...**
Cited by 1428 Related articles All 103 versions Cite Save

[BOOK] Computer arithmetic: algorithms and hardware designs
B Parhami - 2009 - dl.acm.org
... processing. Using a unified and consistent framework, the text begins with number **representation** and proceeds through basic arithmetic operations, **floating-point** arithmetic, and function evaluation methods. Later chapters **...**
Cited by 1303 Related articles All 2 versions Cite Save



ACM Computing Surveys, Vol 23, No 1, March 1991

What Every Computer Scientist Should Know About Floating-Point Arithmetic

DAVID GOLDBERG

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304

Floating-point arithmetic is considered an esotoric subject by many people. This is rather surprising, because floating-point is ubiquitous in computer systems: Almost every language has a floating-point datatype; computers from PCs to supercomputers have floating-point accelerators; most compilers will be called upon to compile floating-point algorithms from time to time; and virtually every operating system must respond to floating-point exceptions such as overflow This paper presents a tutorial on the aspects of floating-point that have a direct impact on designers of computer systems. It begins with background on floating-point representation and rounding error, continues with a discussion of the IEEE floating-point standard, and concludes with examples of how computer system builders can better support floating point.

1. Download the .pdf
2. Search into the .pdf for what you want to find

   Example: “bias”

representation. In the case of single precision, where the exponent is stored in 8 bits, the bias is 127 (for double precision it is 1023). What this means is that if $\overline{k}$ is the value of the exponent bits interpreted as an unsigned integer, then the exponent of the floating-point number is $\overline{k} - 127$. This is often called the *biased exponent* to distinguish from the unbiased exponent $\overline{k}$. An advantage of biased representation is that nonnegative flouting-point numbers can be treated as integers for comparison purposes.

The Problem (example)

Compare 2.5 and 0.25. Which is larger?

$$2.5 = 2^1 + 2^{-1} = 10.1 \times 2^0 = 1.01 \times 2^1$$

$$0.25 = 2^{-2} = 0.01 \times 2^0 = 1.0 \times 2^{-2}$$

Representation with exponent in two's-complement format

```
2.5 = 0 00000001 01000000000000000000000
 -2 = 11111110
0.25 = 0 11111110 00000000000000000000000
```

Representation with exponent in bias format

biasedexponent – 127 = 1 → biasedexponent = 128

```
2.5 = 0 10000000 01000000000000000000000
```

biasedexponent – 127 = -2 → biasedexponent = 125

```
0.25 = 0 01111101 00000000000000000000000
```

The Problem (example)

Compare 2.5 and 0.25. Which is larger?

$$2.5 = 2^1 + 2^{-1} = 10.1 \times 2^0 = 1.01 \times 2^1$$

$$0.25 = 2^{-2} = 0.01 \times 2^0 = 1.0 \times 2^{-2}$$

Representation with exponent in two's-complement format

```
 2.5 = 0 00000001 01000000000000000000000
```

Using integer comparison: 2.5 < 0.25 ✗

```
0.25 = 0 11111110 00000000000000000000000
```

Representation with exponent in biased format

```
 2.5 = 0 10000000 01000000000000000000000
```

Using integer comparison: 0.25 < 2.5 ✓

```
0.25 = 0 01111101 00000000000000000000000
```

Floating Point Representation (example)

1	8	23
S	biasedexponent	fraction

$$N = \begin{cases} (-1)^S \times 1.fraction \times 2^{biasedexponent-127}, & 1 \le biasedexponent \le 254 \\ (-1)^S \times 0.fraction \times 2^{-126}, & biasedexponent = 0 \end{cases}$$

What is the decimal value of the following floating-point number?

$$0\underbrace{100\ 0001\ 1}_{\text{biasedexponent}}001\ 0100\ 0000\ 0000\ 0000\ 0000$$

$$biasedexponent = 2^7 + 2^1 + 2^0 = 128 + 2 + 1 = 131$$

$$actualexponent = biasedexponent - 127 = 131 - 127 = 4$$

$$N = (-1)^0 \times 1.00101_2 \times 2^{131-127} = 1.00101_2 \times 2^4 = 10010.1_2$$

$$N = 2^4 + 2^1 + 2^{-1} = 16 + 2 + \frac{1}{2} = 18.5$$

Floating Point Representation (example)

1	8	23
S	biasedexponent	fraction

$$N = \begin{cases} (-1)^S \times 1.fraction \times 2^{biasedexponent-127}, & 1 \le biasedexponent \le 254 \\ (-1)^S \times 0.fraction \times 2^{-126}, & biasedexponent = 0 \end{cases}$$

What is the largest number that can be represented in 32-bit floating point using the IEEE 754 format above?

$$\mathtt{0}\underbrace{\mathtt{111\ 1111\ 0}}_{\text{biasedexponent}}\mathtt{111\ 1111\ 1111\ 1111\ 1111\ 1111}$$

$$biasedexponent = 2^8 - 2^1 = 256 - 2 = 254$$

$$fraction = 1 \times 2^{-1} + 1 \times 2^{-2} + \cdots + 1 \times 2^{-22} + 1 \times 2^{-23}$$

$$fraction = 1 \times 2^0 - 1 \times 2^{-23} = 1 - \frac{1}{2^{23}} = 1 - \frac{1}{1024 \times 1024 \times 8} = 0.99999988079$$

Floating Point Representation (example)

1	8	23
S	biasedexponent	fraction

$$N = \begin{cases} (-1)^S \times 1.fraction \times 2^{biasedexponent-127}, & 1 \leq biasedexponent \leq 254 \\ (-1)^S \times 0.fraction \times 2^{-126}, & biasedexponent = 0 \end{cases}$$

What is the largest number that can be represented in 32-bit floating point using the IEEE 754 format above?

$$0\underbrace{111\ 1111\ 0}_{\text{biasedexponent}}111\ 1111\ 1111\ 1111\ 1111\ 1111$$

$$actualexponent = 254 - 127 = 127$$

$$fraction = 0.99999988079$$

$$N = (-1)^0 \times 1.99999988079_{10} \times 2^{127} \approx 2^{128}$$

Floating Point Representation (example)

1	8	23
S	biasedexponent	fraction

$$N = \begin{cases} (-1)^S \times 1.fraction \times 2^{biasedexponent-127}, & 1 \le biasedexponent \le 254 \\ (-1)^S \times 0.fraction \times 2^{-126}, & biasedexponent = 0 \end{cases}$$

What is the smallest non-negative number that can be represented in 32-bit floating point using the IEEE 754 format above?

$$0\underbrace{000\ 0000\ 0}_{biasedexponent}000\ 0000\ 0000\ 0000\ 0000\ 0001$$

$$actualexponent = 0 - 126 = -126$$ - note the special rule above for biasedexponent = 0

$$fraction = 1 \times 2^{-23}$$

$$N = (-1)^0 \times 2^{-23} \times 2^{-126} \approx 2^{-149}$$

Special Floating Point Representation

The 8-bit field of the biasedexponent can represent numbers from 0 to 255

What is the value represented when the exponent is 255 (ie. 11111111_2)?

biasedexponent = 255 = 11111111_2 indicates a special value

- biasedexponent = 255 and fraction = 0:
  - the value represented is ±infinity
- biasedexponent = 255 and fraction ≠ 0:
  - the value represented is Not a Number (NaN)

Double Precision

32-bit floating-point representation is usually called single-precision representation

A double-precision floating-point representation requires 64 bits:

- 1 sign bit
- 11 bits for exponent
- 52 bits for fraction (also called significand)

A half-precision floating-point representation requires 16 bits:

- 1 sign bit
- 5 bits for exponent
- 10 bits for fraction (also called significand)

ElectronicDesign.

TECHNOLOGIES > EMBEDDED

Floating-Point Formats in the World of Machine Learning

Sept. 9, 2022

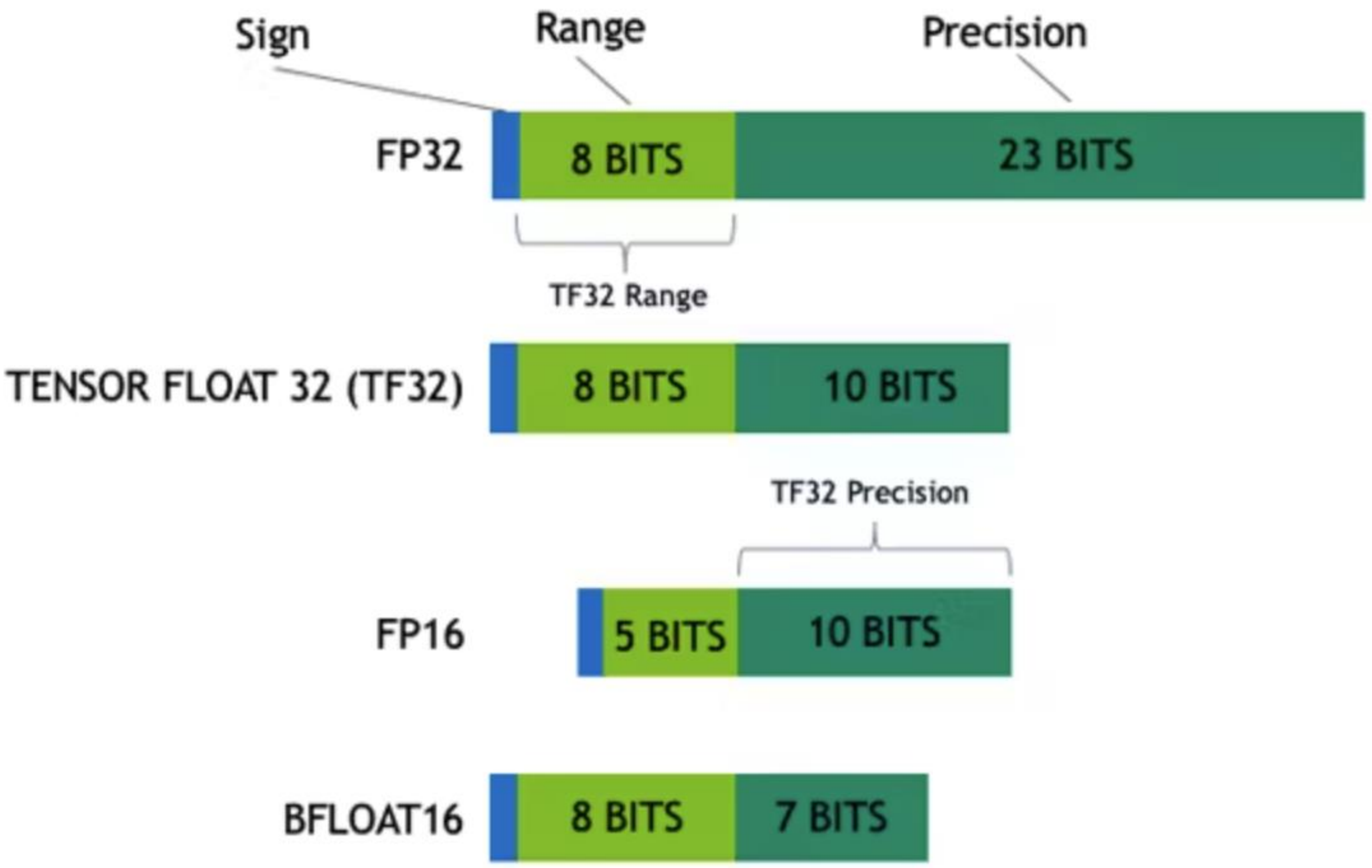
Different floating-point formats allow machine-learning systems to operate more efficiently and use less space.

Cabe Atwell

Related To: Electronic Design

https://www.electronicdesign.com/technologies/embedded/article/21250407/electronic-design-floatingpoint-formats-in-the-world-of-machine-learning

Floating Point Formats


Sign
Range
Precision
FP32
8 BITS
23 BITS
TF32 Range
TENSOR FLOAT 32 (TF32)
8 BITS
10 BITS
TF32 Precision
FP16
5 BITS
10 BITS
BFLOAT16
8 BITS
7 BITS

FP8 FORMATS FOR DEEP LEARNING

Paulius Micikevicius, Dusan Stosic, Patrick Judd, John Kamalu, Stuart Oberman, Mohammad Shoeybi, Michael Siu, Hao Wu
NVIDIA
{pauliusm, dstosic, pjudd, jkamalu, soberman, mshoeybi, msiu, skyw}@nvidia.com

Neil Burgess, Sangwon Ha, Richard Grisenthwaite
Arm
{neil.burgess, sangwon.ha, richard.grisenthwaite}@arm.com

Naveen Mellempudi, Marius Cornea, Alexander Heinecke, Pradeep Dubey
Intel
{naveen.k.mellempudi, marius.cornea, alexander.heinecke, pradeep.dubey}@intel.com

Abstract

FP8 is a natural progression for accelerating deep learning training inference beyond the 16-bit formats common in modern processors. In this paper we propose an 8-bit floating point (FP8) binary interchange format consisting of two encodings - E4M3 (4-bit exponent and 3-bit mantissa) and E5M2 (5-bit exponent and 2-bit mantissa). While E5M2 follows IEEE 754 conventions for representatio of special values, E4M3's dynamic range is extended by not representing infinities and having only one mantissa bit-pattern for NaNs. We demonstrate the efficacy of the FP8 format on a variety of image and language tasks, effectively matching the result quality achieved by 16-bit training sessions. Our study covers the main modern neural network architectures - CNNs, RNNs, and Transformer-based models, leaving all the hyperparameters unchanged from the 16-bit baseline training sessions. Our training experiments include large, up to 175B parameter, language models. We also examine FP8 post-training-quantization of language models trained using 16-bit formats that resisted fixed point int8 quantization.

Table 1: Details of FP8 Binary Formats

	E4M3	E5M2
Exponent bias	7	15