Floating-Point Numbers

Slides adapted by: Sparsh Mittal

Floating Point

- Representation for non-integral numbers
  - Including very small and very large numbers
- Like scientific notation
  - -2.34×10^{56} ← normalized
  - $+0.002 \times 10^{-4}$ ← not normalized
  - $+987.02 \times 10^{9}$ ← not normalized
- In binary
  - $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
- Types **float** and **double** in C

Floating Point Standard

- Defined by IEEE Standard 754-1985
- Developed in response to divergence of representations
  - ➢Portability issues for scientific code
- Now almost universally adopted
- Most-commonly representations
  - ➢Half precision (16-bit)
  - ➢Single precision (32-bit)-------------**float** in C
  - ➢Double precision (64-bit) ------------**double** in C

IEEE Floating-Point Format

	single: 8 bits double: 11 bits	single: 23 bits double: 52 bits
S	Exponent	Fraction

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit ($0 \Rightarrow$ non-negative, $1 \Rightarrow$ negative)
- Normalize significand: $1.0 \leq |\text{significand}| < 2.0$
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1023

Formula for Bias

Bias = $2^{(NumberOfExpBits-1)}-1$

Why Is It Called Single/Double Precision

- The *precision* indicates the number of decimal digits that are **correct**, that is, without any kind of representation error or approximation. In other words, it indicates how many decimal digits one can **safely** use.
- The number of decimal digits which can be safely used:
- **Single precision**: $\log_{10}(2^{24})$, which is about 7~8 decimal digits
- **Double precision**: $\log_{10}(2^{53})$, which is about 15~16 decimal digits

https://stackoverflow.com/a/42444685/984260

Various formats and their correct digits

Precision Type	Total Bits	Sign	Exponent	Significand	Decimal Digits
Half	16	1	5	10	~3.31
Single	32	1	8	23	~7.22
Double	64	1	11	52	~15.95
Quadruple	128	1	15	112	~34.02
Octuple	256	1	19	236	~71.34

Infinities and NaNs

- Exponent = 111…1, Fraction = 000…0
  - ±Infinity
- Exponent = 111…1, Fraction ≠ 000…0
  - Not-a-Number (NaN)
  - Indicates illegal or undefined result
    - For example, 0.0 / 0.0

Special FP Numbers

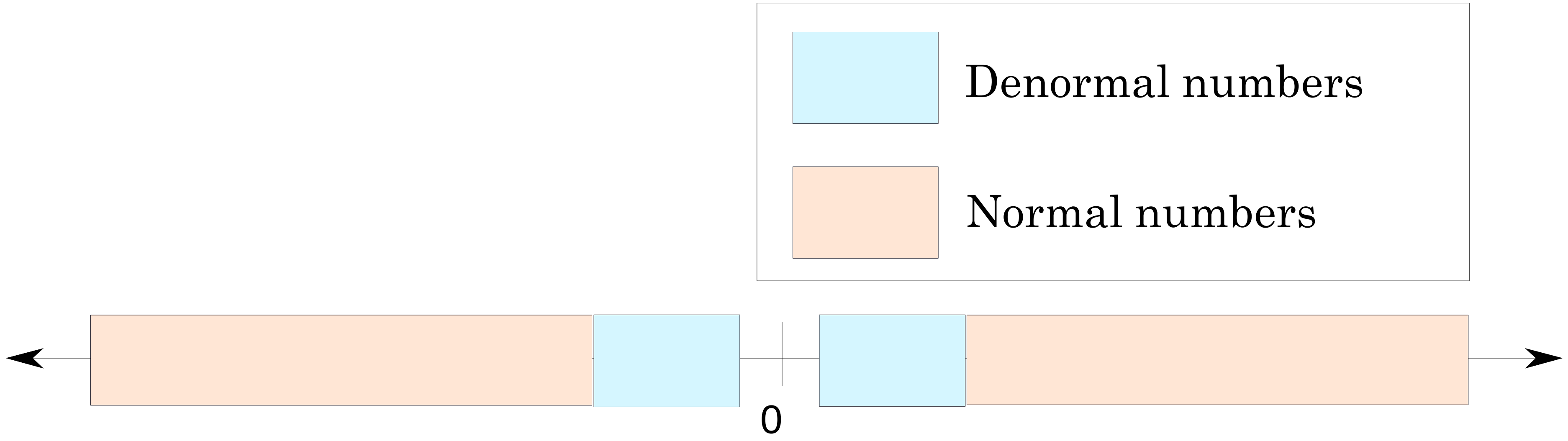
E	M	Value
255	0	∞ if $S = 0$
255	0	$-\infty$ if $S = 1$
255	$\neq 0$	NAN(Not a number)
0	0	0
0	$\neq 0$	Denormal number

This table is for FP32 numbers

* NAN + x= NAN $\qquad$ $1/0 = \infty$
* 0/0 = NAN $\qquad$ $-1/0 = -\infty$
* $\sin^{-1}(5)$ = NAN

E – Exponent, M –Mantissa

Denormal Numbers on Number Line



- In decimal, say $7 * 10^5$ is considered normalized representation, but 0.7*1e6 is not normalized.
- Similarly, in binary, $1.1 * 2^5$ is considered normalized, but $0.11 * 2^6$ is not normalized; it is said “denormal”.

Denormal Numbers

- Exponent = 000...0 $\Rightarrow$ hidden bit is 0

$$x = (-1)^S \times (0 + \text{Fraction}) \times 2^{-126}$$

For FP32

- Smaller than normal numbers
  - Allow for gradual underflow, with diminishing precision

NOTE: For denormal numbers, exponent is NOT 0-bias, but 1-bias. Bias is 127, so we get 1-127 = -126

Denormal Numbers

* Smallest +ve normal number : 2^{-126}
* Largest denormal number :
  * $0.11...11 * 2^{-126} = (1 – 2^{-23})*2^{-126}$
  * $=2^{-126} - 2^{-149}$

Summary representation

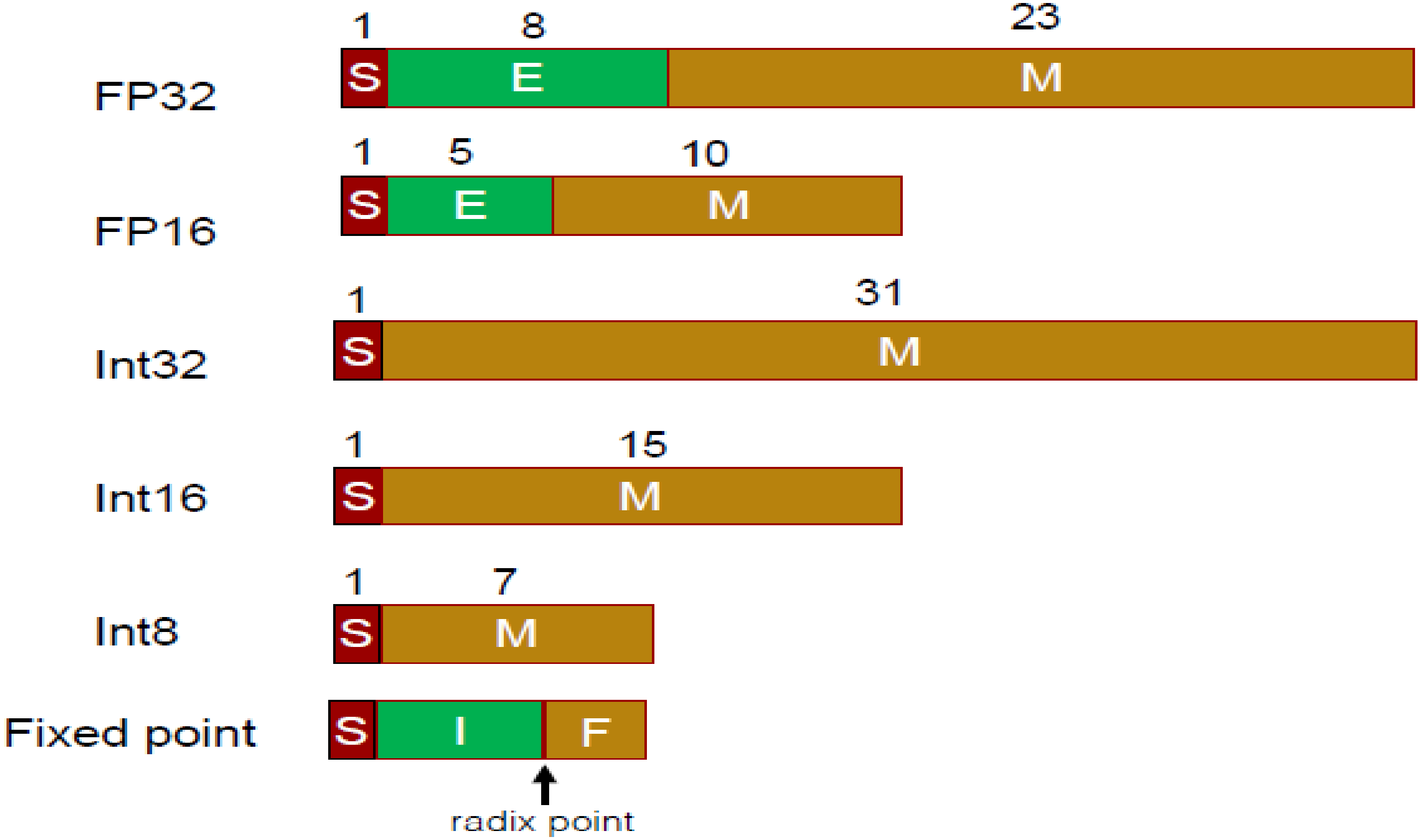
$$
\text{Float value} = \begin{cases} e = \text{all } 1 \text{ bits}, & \begin{cases} f = 0, & (-1)^s \infty, \\ f \neq 0, & \text{NaN of some kind}, \end{cases} \\ e = \text{all } 0 \text{ bits}, & \begin{cases} f = 0, & \begin{cases} s = 0, & \text{"Positive zero"}, \\ s = 1, & \text{"Negative zero"}, \end{cases} \\ f \neq 0, & (-1)^s \times 2^{1-bias} \times f, \end{cases} \\ \text{all other } e, & (-1)^s \times 2^{e-bias} \times (1 + f). \end{cases}
$$

Note: We have to first check whether a number belongs to special cases (0/infinity/NaN/denormal).
If a number does not belong to special case, then, it is taken as a normal number.

Fixed-point format

- **FxP** has a specific number of bits (or digits) reserved for integer and fractional parts, regardless of how large/small the number is. For example:
  With IIIII.FFFFF format, we can show numbers in range [00000.00000, 11111.11111] (binary system)
- **FP**: the number of bits for integer/fractional part is not reserved. Instead, it reserves certain bits for the significand and exponent
- **Int** is similar to FxP, except that Int has no fraction part.
- Sometimes, Int and FxP are used synonoymously

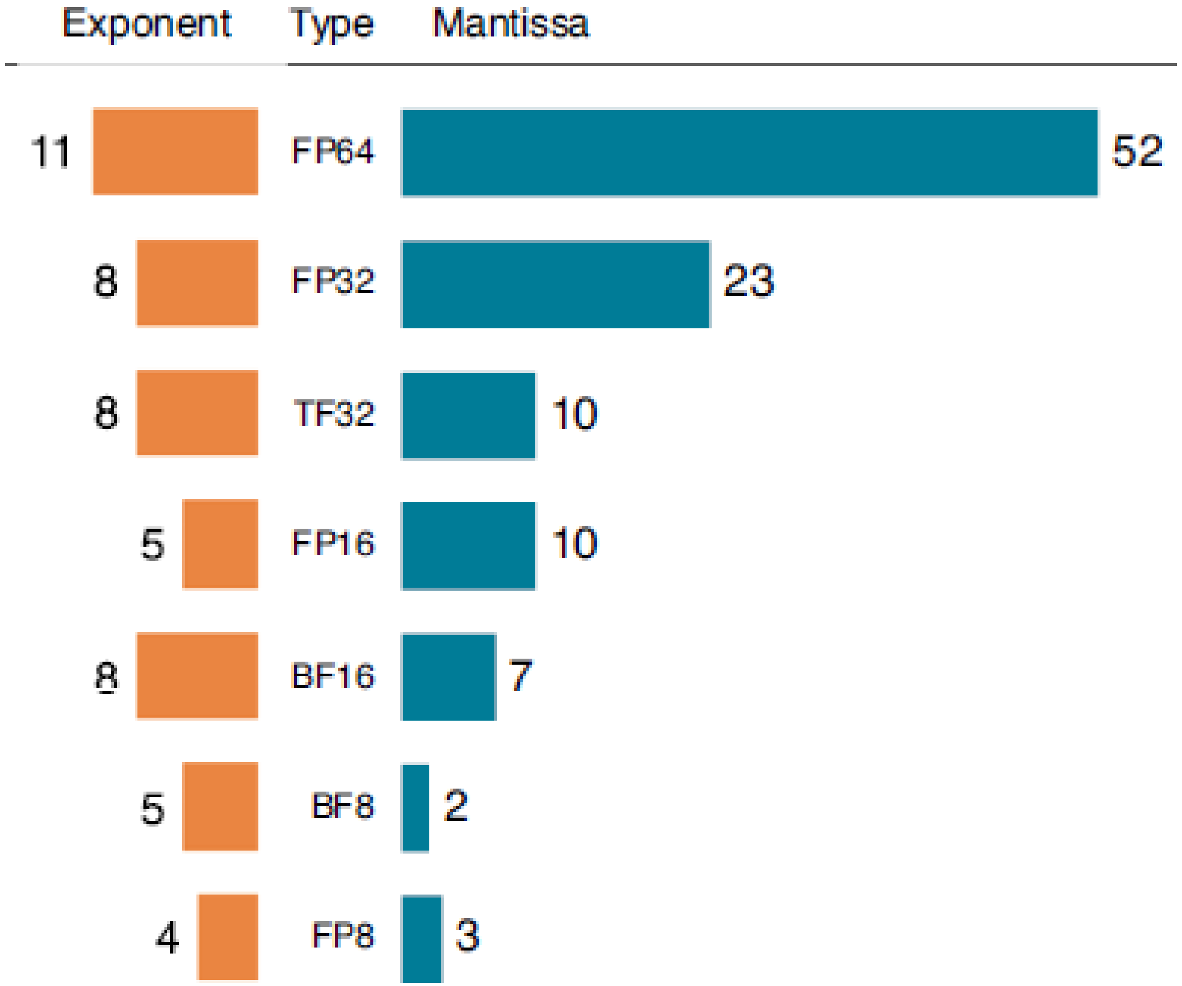
$$(-1)^S \times (1.M) \times 2^E$$

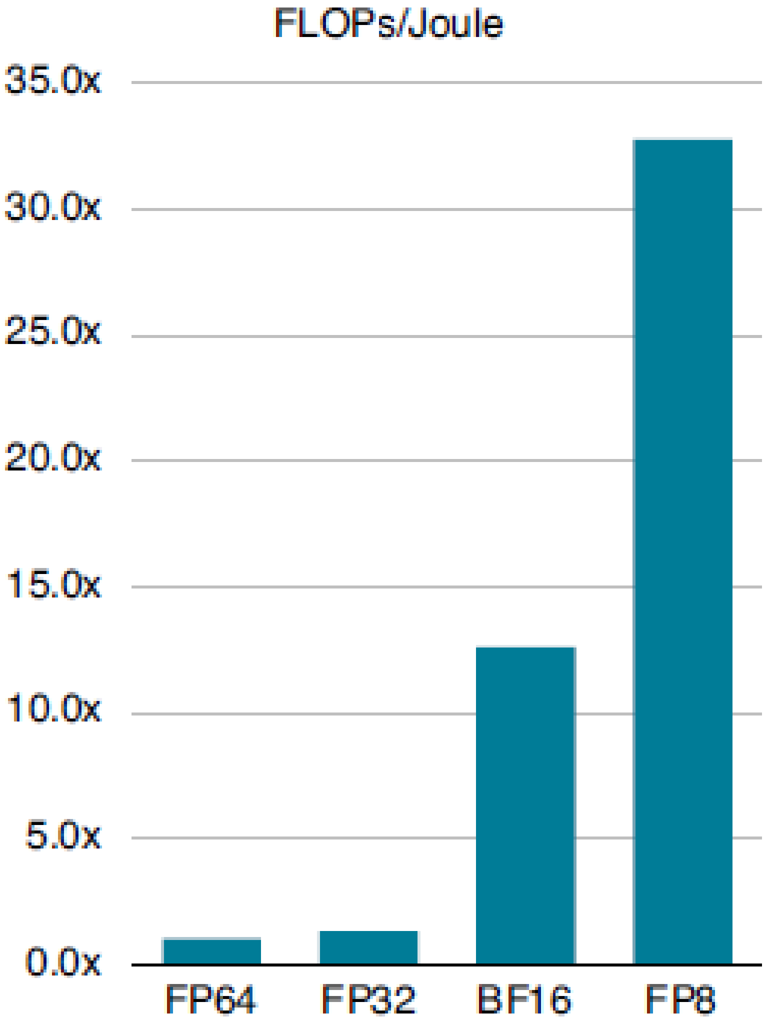


	Range	Error
FP32	10^{-38} - 10^{38}	.000006%
FP16	$6x10^{-5}$ - $6x10^{4}$	.05%
Int32	$0 – 2x10^{9}$	½
Int16	$0 – 6x10^{4}$	½
Int8	0 – 127	½
Fixed point	-	-

Dally, High Performance Hardware for Machine Learning, NIPS'2015

Selected Floating Point Formats and Associated Energy Efficiency





Further Study

- https://blog.demofox.org/2017/11/21/floating-point-precision/
- https://www.h-schmidt.net/FloatConverter/IEEE754.html
- https://stackoverflow.com/questions/4220417/print-binary-representation-of-a-float-number-in-c
- https://moocaholic.medium.com/fp64-fp32-fp16-bfloat16-tf32-and-other-mem bers-of-the-zoo-a1ca7897d407
- https://www.ibm.com/support/pages/single-precision-floating-point-accuracy
- http://www.mimirgames.com/articles/programming/digits-of-pi-needed-for-floating-point-numbers/