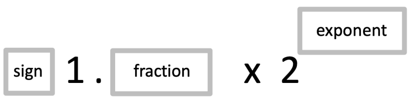
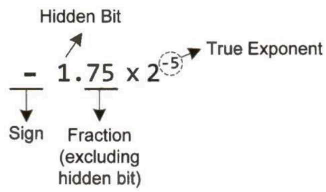
Floating Point Representation

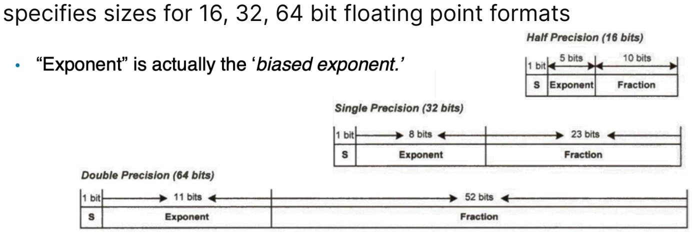
- normalized notation with 3 bit fields
- **implicit one** only one digit before the decimal pt, whose value is ONE which is omitted
  - Sign
  - Fraction (Mantissa)
  - Exponent (binary field holding a biased exponent)





IEEE754 Format

- specifies sizes for 16, 32, 64 bit floating point formats



Biased Exponent

- Biased Exponent = Actual Exponent + Exponent Bias
  - bias for 16bit = 15
  - bias for 32bit = 127
  - bias for 64bit = 1023
- exponent bias = $2^{b-1} - 1$

  $F = (-1)^s \times (1 + fraction) \times 2^{(biased\ exp\ -\ bias)}$

IEEE754 Exceptions

sign	Exponent	Fractional	Number
0 or 1	00…0	00…0	± ZERO
0 or 1	00…0	Any non-zero	Subnormal
0 or 1	11…1	00…0	± infinity
0 or 1	11…1	Any non-zero	NaN (e.g., sqrt(-1))

Underflow & Overflow

- **underflow** - exact result non-zero and smaller than the smallest representable value
  - loss of precision and lead to computational error
- **overflow** - exact result is finite but exceeds largest representable value
- when either detected, hardware returns zero / max + exception signal

Subnormal Numbers

- represent numbers below $\pm 1.18 \times 10^{-38}$ for 32 bit values
  - subnormals extend the range to $\pm 1 \times 10^{-45}$

Smallest/Largest value representable

- Smallest non-zero is exponent 0b0000 0001 with fraction all zeros (23 bits)

$$(-1)^s \times (1+0) \times 2^{1-127} = \pm 2^{-126} \approx 1.18 \times 10^{-38}$$

- Why is this the smallest value?
  - Exponent = 0b0000 0000 indicates a subnormal number.

- So, the smallest non-zero, **non-subnormal** number is 0b0000 0001 0000… 00

- Largest value is exponent 0b1111 111**0** with fraction all ones (23 bits)

$$(-1)^s \times (1+(1-2^{-23})) \times 2^{254-127} = \pm(2^{128}-2^{104}) \approx 3.40 \times 10^{38}$$

- Similarly, why is the largest exponent not 0b1111 111**1** ?
  - IEEE754 reserves this for ±∞ and ±NaN

- So, our largest exponent is 0b1111 1110 = 254
- We cannot use exponents of 0b0000 0000 or 0b1111 1111 for computations.

FP Resolution

- FP numbers are not distributed uniformly across range
  - fixed point, resolution is the same across the entire range
  - fp, resolution increases as number gets bigger

  Let's see two cases for 32-bit IEEE-754
  - E.g. 1) 10 => 0x4120 0000
    - Adding 1 to this number gives 10.000001
    - By changing LSB, value changes by 10^{-6}.

  - E.g. 2) 10000 => 0x461C 4000
    - Adding 1 to this number gives 10000.001
    - By changing LSB, value changes by 10^{-3}.

FP Rounding

- finite bits; computation produces more bits than that can be stored, must truncate values
  - rounding modes - nearest value, towards 0, towards $\pm\infty$
  - use truncation

Rounding Rule	Data	Rounded Result
Nearest	+0.123456	+0.12346
	-0.123456	-0.12346
Truncate	+0.123456	+0.12345
	-0.123456	-0.12345
Rounding up	+0.123456	+0.12346
	-0.123456	-0.12345
Rounding down	+0.123456	+0.12345
	-0.123456	-0.12346

FP Operations

Multiplication

- first multiply decimal parts (mantissa in fp)
- add the exponents
- normalize (standard notation) result to begin with 1

① Calculate SR = S1 XOR S2, to get the sign of the result.

② Add exponents ER = E1 + E2

③ Multiply mantissas MR = M1 * M2 and truncate

④ Normalize Mantissa (and adjust ER if needed)

⑤ Check for special conditions (overflow, underflow, special values etc.)

S1	E1	M1
	*	
S2	E2	M2
	=	
SR	ER	MR

Addition & Subtraction

① If E1 != E2, adjust the larger to match the smaller.

② Add the mantissas.

③ Normalize result mantissa

④ Check for special conditions (overflow, underflow, special values etc.)

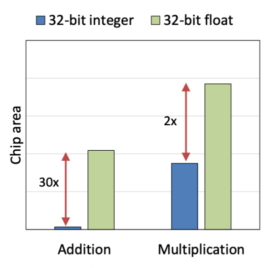
S1	E1	M1
	+	
S2	E2	M2
	=	
SR	ER	MR

Does your hardware support it? What if it doesn't?
- Without dedicated FP hardware, your code will default to 'software floating point'.
- Table on the right shows the number of cycles taken on an ARM CPU to do FP operations with and without hardware support.
- If so, FP operations will be a **lot** slower!

Operation	With FPU	Without FPU
ADD	1	102
SUB	1	108
MUL	1	166
DIV	14	475

Drawbacks of Float

- Despite many advantages of floats over fixed point, the biggest disadvantage is area.
- Compared to a 32-bit integer adder, a 32-bit FP adder costs 30x the chip area.
- A multiplier costs 2x as much as an integer multiplier.
- If you **do** have FPU hardware, how can things **still** go wrong?



Floating Point Operations

- rounding error can accumulate
  - use doubles
- operations are not commutative
- other formats
  - bfloat16 (S + 8E + 7M)
  - FP8
  - FP4