FLOATING POINT

Floating Point Number Components

$$sign \times mantissa \times base^{exp\_sign \times (exp - bias)}$$

- *sign* (optional) determines the sign of the overall number
- *mantissa* can take many forms
  - unsigned, sign magnitude, 2's complement
  - integer, full fractional < 1, mixed integer.fractional
- *base* is normally fixed in a particular usage and is commonly 2
  - it is not explicitly stored or transmitted
  - other common values include a power-of-2 and 10
- *exp* can also take the same forms
  - unsigned, sign magnitude, 2's complement
  - integer, full fractional < 1, mixed integer.fractional
- *exp_sign* (optional) determines the sign of the exponent
- *bias* (optional) has advantages in enabling faster comparisons between floating point numbers

Floating Point Arithmetic

- Multiplication
  - Relatively straightforward because the *mantissa* and *exponent* can be processed independently
- Addition and Subtraction
  - More complex because the mantissas require *alignment* before addition or subtraction can begin
- All arithmetic operations
  - Require *normalization* after the arithmetic is complete

Floating Point Storage and Transmittal

- $$mantissa * base^{exponent}$$
  $$sign * mantissa * base^{exponent}$$
- We normally never explicitly store or transmit the *base;* only the other values:

  (MMMMMMMM, EEEEE)
  (S_{MAN}, MMMMMMMM, EEEEE)
  (S_{MAN}, MMMMMMMM, S_{EXP}, EEEEE)
- For example, if the exponent is stored as a 4-bit 2's complement integer:

```
00010. * 2^0         (00010,0000)
00101. * 2^(-1)      (00101,1111)
```

Floating Point Example
All 8-bit Unsigned Numbers

a) fixed-point integer
XXXXXXXX.
range [0–255]
resolution or smallest step size = 1

b) fixed-point "4.4 format" fractional
XXXX.XXXX
range [0 – 15 15/16] = [0 – 15.9375]
resolution = 1/16 = 0.0625

c) floating-point 4-bit integer mantissa, 4-bit 2's complement exponent
$[0 - 15] \times 2^{[-8 - +7]}$

When *exp* = –8	range [0 – 0.0586]	resolution 0.0039
When *exp* = 0	range [0 – 15]	resolution 1
When *exp* = +7	range [0 – 1920]	resolution 128

Normalization example with an Integer Mantissa

- *Normalized* floating point numbers contain no extra (useful) bits at the MSB end of the mantissa
  - Examples for 2.75_{10} with an **unsigned** 5-bit **integer** mantissa:

    `00010. * 2^0` not normalized, or "denormalized"
    `00101. * 2^(-1)` not normalized, or "denormalized"
    `01011. * 2^(-2)` not normalized, or "denormalized"
    `10110. * 2^(-3)` normalized
  - A good starting point in such problems is to first consider the original value in fixed point which is **`10.11`** or **`010.1100`** etc., in this case
  - To keep this example simple, bits in the original value of 2.75 that do not fit into the mantissa (i.e., the first two cases) are dropped or truncated, which is not as accurate as rounding

Normalization example with a 0.5 Fractional Mantissa

- *Normalized* floating point numbers contain no extra (useful) bits at the MSB of the mantissa
  - Examples for 2.75_{10} with an **unsigned** 5-bit **"0.5 format" fractional** mantissa:

```
.00010 * 2^(+5)        not normalized, or "denormalized"
.00101 * 2^(+4)        not normalized, or "denormalized"
.01011 * 2^(+3)        not normalized, or "denormalized"
.10110 * 2^(+2)        normalized
```

  - If the exponent is stored as a 4-bit 2's complement integer, these 4 examples would be stored as:

```
(MMMMM,EEEE)
(00010,0101)
(00101,0100)
(01011,0011)
(10110,0010)
```

Normalization example with a 2's Complement Mantissa

- *Normalized* floating point numbers contain no extra (useful) bits at the MSB end of the mantissa
  - Examples for $+2.75_{10}$ with a **2's complement** 5-bit **integer** mantissa:

`00010. * 2^0`	not normalized, or "denormalized"
`00101. * 2^(-1)`	not normalized, or "denormalized"
`01011. * 2^(-2)`	normalized
`10110. * 2^(-3)`	Broken! Lost sign bit

Normalization example with a 2's Complement Mantissa

- *Normalized* floating point numbers contain no extra (useful) bits at the MSB end of the mantissa
  - Examples for **–2.75$_{10}$** with a 2's complement 5-bit integer mantissa
  - First calculate the value in fixed point:
    `–2.75`$_{10}$ `= 101.010 = (-4)+(1)+(1/4)`
  - Then calculate cases with various exponent values:

`11101. * 2^0`	not normalized, or "denormalized"
`11010. * 2^(-1)`	not normalized, or "denormalized"
`10101. * 2^(-2)`	normalized
`01010. * 2^(-3)`	Broken! Lost sign bit