The Character of Binary8 Floating-Point Formats

Jeffrey Sarnoff

2023-Oct-17 (v0.5.2)

Early pre-release for review and corrections. This version may contain errors or inaccuracies.

This development works with binary8p3, binary8p4, binary8p5, binary8p6, binary8p7. The remaining binary8 formats (binary8p1, binary8p2, binary8p8) *are not fully amenable to the characterization used.*

Bits spanned

k is the number of bits required of the binary interchange format (always 8 for binary8s)

```
                    # an octothorpe '#' starts an inline comment
  const k   = 8     # the bits required to encode binary8 values
  const k_m = 7     # the bits required to encode binary8 magnitudes[1] (k - 1)
```

All binary8 formats have 256 == 2^8 == 2^k unique encodings. Binary8 formats are given by their precision.

Each binary8 format maps the same 256 encodings (0x00..0xff) onto its own set of 256 values. Each of these values is unique within its value set. Distinct binary8 formats may share some of their values. Where subnormal or normal values are shared by formats, they are mapped from distinct encodings.

Precision

p is the precision of the format in bits (p includes the implicit bit of the significand). The number of bits explicitly stored for the significand is precision – 1. This has to be >= 0; so the smallest precision is 1. The largest precision for binary8 formats is 8, which is used with a 7 bit wide explicit significand. There is always a sign bit, so 7 bits is the most available for other purposes.

The names binary8p{ **p** } select 8-bit binary floating-point interchange formats by their precisions { **p** }.

[1] k_m is not a formal parameter from 754. I use it for simplifying other parametric expressions.

Kinds of Value

Values are either Normal, Subnormal or Special. These are the 4 special values {0, +Inf, -Inf, NaN}. Special values are neither normal nor subnormal as 754 defines those terms. There are 252 (256 - 4) ordinary values, each ordinary value is either a normal or a subnormal value.

```
  const n_values        = 2^k                         # 256, the number of encodings
  const n_specials      = 4                           # the number of non[sub]normals
  const n_ordinaries    = n_values - n_specials       # 252, the number of [sub]normals
```

signed and unsigned values

Binary8 formats have n_values. Half of them are encoded with the msb[2] clear (nonnegative values), and half are encoded with the msb set (negative values). Note that half of our four special values are nonnegative and half are negative. Also, half of our ordinary values are nonnegative and half are negative.

With binary8 formats, zero is neither negative nor positive; consider zero unsigned until there is a compelling reason to ascribe some oriented sense, then use the positive sense. For example, evaluating divide(0x01, 0x00) is +Inf rather than -Inf or NaN.

```
  const n_pos_ordinaries = n_ordinaries >> 1          # 126
  const n_pos_numbers    = n_pos_ordinaries + 1       # 127, includes +Inf
  const n_nonneg_numbers = n_pos_numbers + 1          # 128, includes Zero
```

magnitudes

Ordinary values are signed values. They are positive (the msb is 0b0) or negative (the msb is 0b1).

There are (n_ordinaries >> 1) positive signed values and (n_ordinaries >> 1) negative signed values.[3]

We may use positive numbers to serve as nonzero magnitudes. Note that Zero is not a magnitude since it is neither normal nor subnormal. This approach works by relying on the msb of a positive value to be 0b0.

```
  const n_magnitudes = n_ordinaries >> 1       # 126, half the number of [sub]normals
```

Parameters

IEEE Std 754-2019

w	the bitwidth of exponent field	derived from k, p	8-p (== k-p)
t	the explicit significand bit count	derived from p	p-1
emax	the max unbiased exponent value	defined from k_m, p	2^(7 - p) - 1

[2] *msb* is an acronym for *most significant bit*

[3] With positive evens, div by 2 is (poseven >> 1). This form abstracts away language specifics in (poseven / 2).

bias	the exponent bias, emax(p) + 1	defined from k_m, p	2^(7 - p)

mneumonics

```
sig_bits(p) = p                             # the precision
frc_bits(p) = p - 1                         # the fraction part of the significand

exp_bits(p) = 8 - p                         # k - p, exponent bitfield width
exp_bias(p) = 2^(7 - p)                     # the exponent bias, 2^(exp_bits(p) - 1)
exp_max(p)  = 2^(7 - p) - 1                 # max unbiased exponent, exp_bias(p) - 1
```

Counts

Counting Significands

The number of unique significands for normal values of a binary8 format is 2^precision. Subnormal values do not include the zero-valued significand (that is used to encode Zero and NaN).

```
n_normal_significands(p)        = 2^(p)
n_normal_significand_mags(p)    = 2^(p - 1)

n_subnormal_significands(p )    = 2^(p) - 2      # n_normal_significands(p) - 2
n_subnormal_significand_mags(p) = 2^(p-1) - 1    # n_subnormal_significands(p) >> 1
```

Counting Values

There is one subnormal value for each subnormal significand. There is exactly one subnormal exponent; subnormal significands do not cycle. Unlike normal values, each subnormal significand occurs once.

```
n_subnormals(p)     = 2^(p) - 2                  # n_normals(p) - 2
n_subnormal_mags(p) = 2^(p - 1) - 1              # n_subnormals(p) >> 1
```

The number of normal values is given from n_ordinaries and the number of subnormal values. There are half as many normal magnitudes as there are normal values.

```
n_normals(p)     = n_ordinaries - n_subnormals(p)
n_normal_mags(p) = n_normals(p) >> 1
```

Extremal Magnitudes

```
min_normal(p)    = recip( 2^(8 - p) )
min_subnormal(p) = recip( 2^(2p - 3) )          # 8 * recip( 2^(2*p) )

max_subnormal(p) = min_normal(p) - min_subnormal(p)
max_normal(p)    = (2^p - 2) * 2^(2^(7-p) - p)
```