CSC 252: Computer Organization Spring 2018: Lecture 4

Instructor: Yuhao Zhu

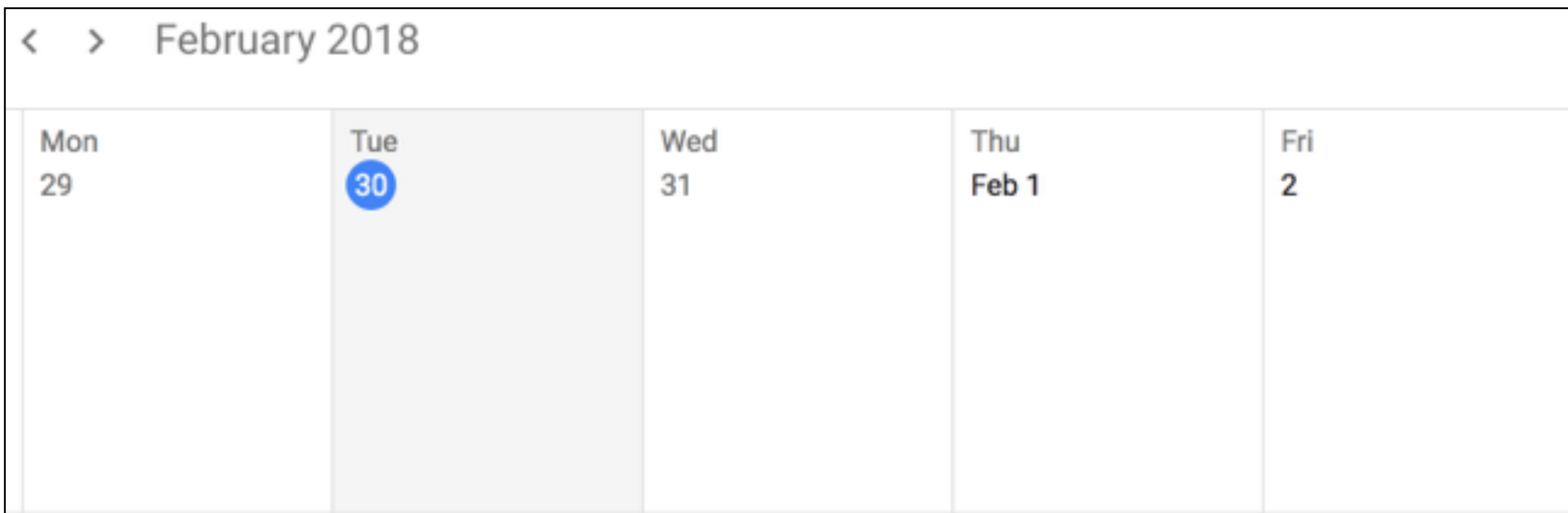
Department of Computer Science
University of Rochester

Action Items:

- **Assignment 1 due Feb. 2, midnight**

Announcement

- Programming Assignment 1 is out
  - Due on **Feb 2, 11:59 PM**
  - You have 3 slip days
  - Try to submit once to make sure you can submit
  - We count only the latest submission before the deadline



Announcement

- Programming Assignment 1 is out
  - Due on **Feb 2, 11:59 PM**
  - You have 3 slip days
  - Try to submit once to make sure you can submit
  - We count only the latest submission before the deadline
- TAs are better positioned to answer questions regarding assignments

Previously in 252…

Previously in 252…

- Signed vs. Unsigned Integer
    - Integer is a special case of fixed-point
    - Fractions can also be represented in fixed-point

Previously in 252…

- Signed vs. Unsigned Integer
  - Integer is a special case of fixed-point
  - Fractions can also be represented in fixed-point
- Least significant bit (byte)
  - Bit (byte) that is least significant to the numerical value of the bit stream — always the rightmost!
  - Has nothing to do with which endianness you choose

Previously in 252…

- Signed vs. Unsigned Integer
  - Integer is a special case of fixed-point
  - Fractions can also be represented in fixed-point
- Least significant bit (byte)
  - Bit (byte) that is least significant to the numerical value of the bit stream — always the rightmost!
  - Has nothing to do with which endianness you choose

10100011

Previously in 252…

- Signed vs. Unsigned Integer
  - Integer is a special case of fixed-point
  - Fractions can also be represented in fixed-point
- Least significant bit (byte)
  - Bit (byte) that is least significant to the numerical value of the bit stream — always the rightmost!
  - Has nothing to do with which endianness you choose

10100011 Least significant bit

Previously in 252…

- Signed vs. Unsigned Integer
  - Integer is a special case of fixed-point
  - Fractions can also be represented in fixed-point
- Least significant bit (byte)
  - Bit (byte) that is least significant to the numerical value of the bit stream — always the rightmost!
  - Has nothing to do with which endianness you choose

Most significant bit 10100011 Least significant bit

Previously in 252…

- Signed vs. Unsigned Integer
  - Integer is a special case of fixed-point
  - Fractions can also be represented in fixed-point
- Least significant bit (byte)
  - Bit (byte) that is least significant to the numerical value of the bit stream — always the rightmost!
  - Has nothing to do with which endianness you choose

Most significant bit 10100011 Least significant bit

DEADBEEF

Previously in 252…

- Signed vs. Unsigned Integer
  - Integer is a special case of fixed-point
  - Fractions can also be represented in fixed-point
- Least significant bit (byte)
  - Bit (byte) that is least significant to the numerical value of the bit stream — always the rightmost!
  - Has nothing to do with which endianness you choose

Most significant bit 10100011 Least significant bit

Most significant byte DEADBEEF

Previously in 252…

- Signed vs. Unsigned Integer
  - Integer is a special case of fixed-point
  - Fractions can also be represented in fixed-point
- Least significant bit (byte)
  - Bit (byte) that is least significant to the numerical value of the bit stream — always the rightmost!
  - Has nothing to do with which endianness you choose

Most significant bit 10100011 Least significant bit

Most significant byte DEADBEEF Least significant byte

Previously in 252…

Google "Hexspeak"

- Signed vs. Unsigned Integer
  - Integer is a special case of fixed-point
  - Fractions can also be represented in fixed-point
- Least significant bit (byte)
  - Bit (byte) that is least significant to the numerical value of the bit stream — always the rightmost!
  - Has nothing to do with which endianness you choose

Most significant bit 10100011 Least significant bit

Most significant byte DEADBEEF Least significant byte

Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + 2*10^0 + 4*10^{-1} + 5*10^{-2}$$

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = \boxed{1*10^1} + 2*10^0 + 4*10^{-1} + 5*10^{-2}$$

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + \boxed{2*10^0} + 4*10^{-1} + 5*10^{-2}$$

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + 2*10^0 + \boxed{4*10^{-1}} + 5*10^{-2}$$

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + 2*10^0 + 4*10^{-1} + \boxed{5*10^{-2}}$$

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + 2*10^0 + 4*10^{-1} + 5*10^{-2}$$

$$10.01_2 = 1*2^1 + 0*2^0 + 0*2^{-1} + 1*2^{-2}$$

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + 2*10^0 + 4*10^{-1} + 5*10^{-2}$$

$$10.01_2 = \boxed{1*2^1} + 0*2^0 + 0*2^{-1} + 1*2^{-2}$$

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + 2*10^0 + 4*10^{-1} + 5*10^{-2}$$

$$10.01_2 = 1*2^1 + \boxed{0*2^0} + 0*2^{-1} + 1*2^{-2}$$

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + 2*10^0 + 4*10^{-1} + 5*10^{-2}$$

$$10.01_2 = 1*2^1 + 0*2^0 + \boxed{0*2^{-1}} + 1*2^{-2}$$

Can We Represent Fractions in Binary?

- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + 2*10^0 + 4*10^{-1} + 5*10^{-2}$$

$$10.01_2 = 1*2^1 + 0*2^0 + 0*2^{-1} + \boxed{1*2^{-2}}$$

Can We Represent Fractions in Binary?

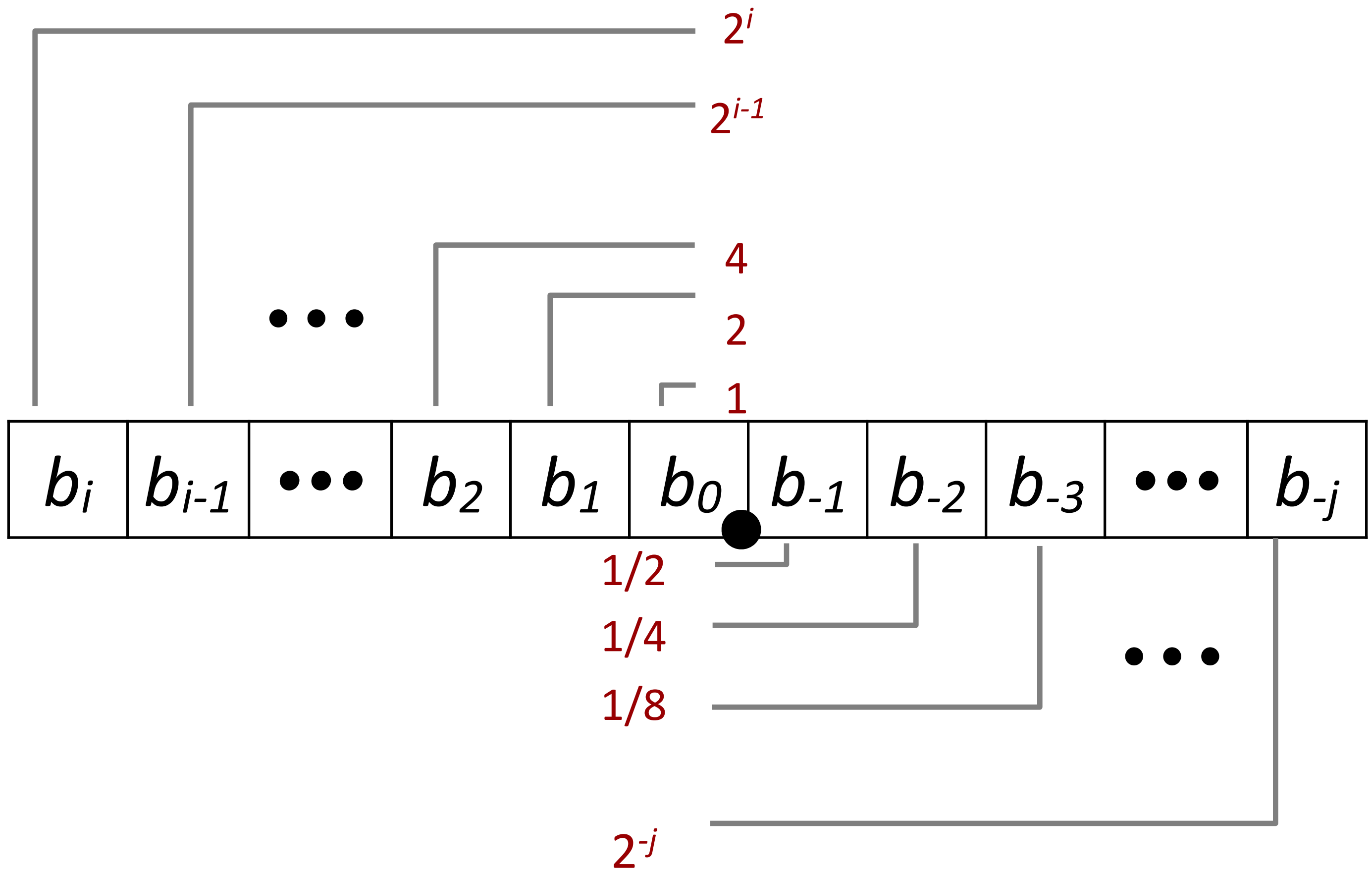
- What does 10.01_2 mean?
  - C.f., Decimal

$$12.45 = 1*10^1 + 2*10^0 + 4*10^{-1} + 5*10^{-2}$$

$$10.01_2 = 1*2^1 + 0*2^0 + 0*2^{-1} + 1*2^{-2}$$

$$= 2.25_{10}$$

Fractional Binary Numbers



Fractional Binary Numbers: Examples

Decimal Value	**Binary Representation**
5 3/4	101.11
2 7/8	10.111
1 7/16	1.0111

Fractional Binary Numbers: Examples

Decimal Value	Binary Representation
5 3/4	101.11
2 7/8	10.111
1 7/16	1.0111

Exact Same Raw Bit Stream!

Fractional Binary Numbers: Examples

Decimal Value	**Binary Representation**
5 3/4	101.11
2 7/8	10.111
1 7/16	1.0111

Exact Same Raw Bit Stream!

- We would need to remember:
  - The raw bit stream
  - Where the binary point is

Fractional Binary Numbers: Examples

Decimal Value	Binary Representation
5 3/4	101.11
2 7/8	10.111
1 7/16	1.0111

Exact Same Raw Bit Stream!

- We would need to remember:
  - The raw bit stream
  - Where the binary point is
- Makes calculations (e.g. addition) hard, and not very elegant
  - Need to first align numbers according to the binary point

Fixed-Point Representation

Fixed-Point Representation

- Binary point stays fixed

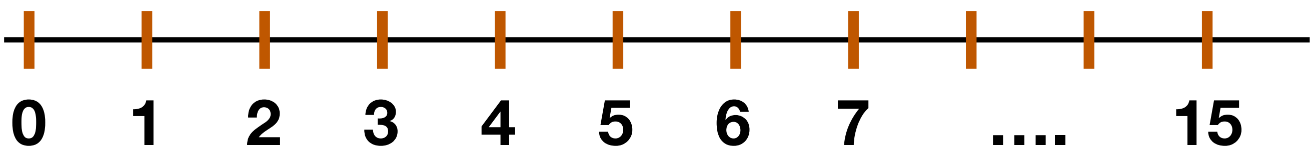
Fixed-Point Representation

- Binary point stays fixed

Decimal	Binary
0	0000.
1	0001.
2	0010.
3	0011.
4	0100.
5	0101.
6	0110.
7	0111.
8	1000.
9	1001.
10	1010.
11	1011.
12	1100.
13	1101.
14	1110.
15	1111.

Fixed-Point Representation

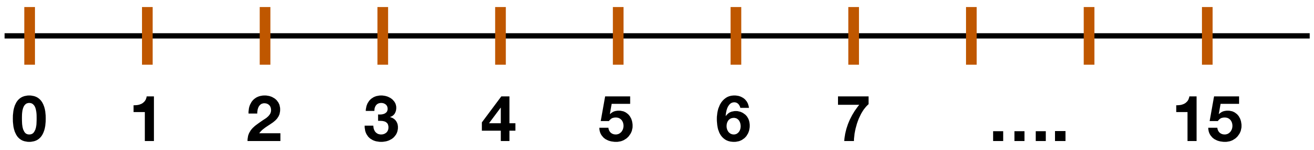
- Binary point stays fixed



Decimal	Binary
0	0000.
1	0001.
2	0010.
3	0011.
4	0100.
5	0101.
6	0110.
7	0111.
8	1000.
9	1001.
10	1010.
11	1011.
12	1100.
13	1101.
14	1110.
15	1111.

Fixed-Point Representation

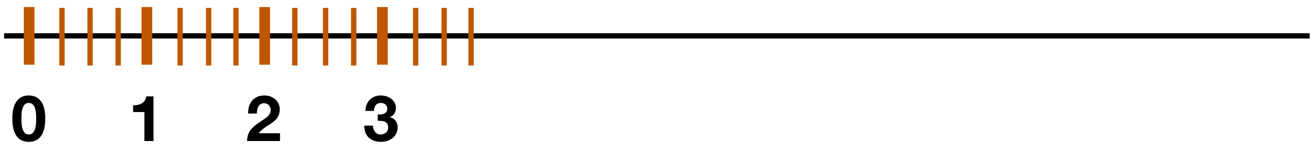
- Binary point stays fixed



Decimal	Binary
0	00.00
0.25	00.01
0.5	00.10
0.75	00.11
1	01.00
1.25	01.01
1.5	01.10
1.75	01.11
2	10.00
2.25	10.01
2.5	10.10
2.75	10.11
3	11.00
3.25	11.01
3.5	11.10
3.75	11.11

Fixed-Point Representation

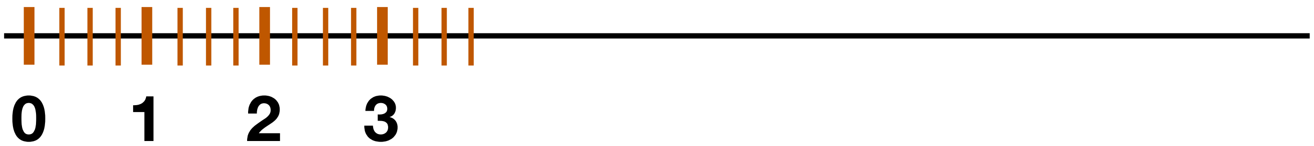
- Binary point stays fixed



Decimal	Binary
0	00.00
0.25	00.01
0.5	00.10
0.75	00.11
1	01.00
1.25	01.01
1.5	01.10
1.75	01.11
2	10.00
2.25	10.01
2.5	10.10
2.75	10.11
3	11.00
3.25	11.01
3.5	11.10
3.75	11.11

Fixed-Point Representation

- Binary point stays fixed
- Fixed interval between representable numbers
  - Each bit represents 0.25_{10}



Decimal	Binary
0	00.00
0.25	00.01
0.5	00.10
0.75	00.11
1	01.00
1.25	01.01
1.5	01.10
1.75	01.11
2	10.00
2.25	10.01
2.5	10.10
2.75	10.11
3	11.00
3.25	11.01
3.5	11.10
3.75	11.11

Fixed-Point Representation

- Binary point stays fixed
- Fixed interval between representable numbers
  - Each bit represents 0.25_{10}

- Still need to remember the binary point, but just once for all numbers

Decimal	Binary
0	00.00
0.25	00.01
0.5	00.10
0.75	00.11
1	01.00
1.25	01.01
1.5	01.10
1.75	01.11
2	10.00
2.25	10.01
2.5	10.10
2.75	10.11
3	11.00
3.25	11.01
3.5	11.10
3.75	11.11

Fixed-Point Representation

- Binary point stays fixed
- Fixed interval between representable numbers
  - Each bit represents 0.25_{10}

- Still need to remember the binary point, but just once for all numbers
- No need to align (already aligned)

Decimal	Binary
0	00.00
0.25	00.01
0.5	00.10
0.75	00.11
1	01.00
1.25	01.01
1.5	01.10
1.75	01.11
2	10.00
2.25	10.01
2.5	10.10
2.75	10.11
3	11.00
3.25	11.01
3.5	11.10
3.75	11.11

Fixed-Point Representation

- Binary point stays fixed
- Fixed interval between representable numbers
  - Each bit represents 0.25_{10}

- Still need to remember the binary point, but just once for all numbers
- No need to align (already aligned)
- C uses fixed-point encoding only for integral data types (**`long`**, **`int`**, **`short`**, etc.)
  - Effectively, implicitly assumes the binary point is at the rightmost

Decimal	Binary
0	00.00
0.25	00.01
0.5	00.10
0.75	00.11
1	01.00
1.25	01.01
1.5	01.10
1.75	01.11
2	10.00
2.25	10.01
2.5	10.10
2.75	10.11
3	11.00
3.25	11.01
3.5	11.10
3.75	11.11

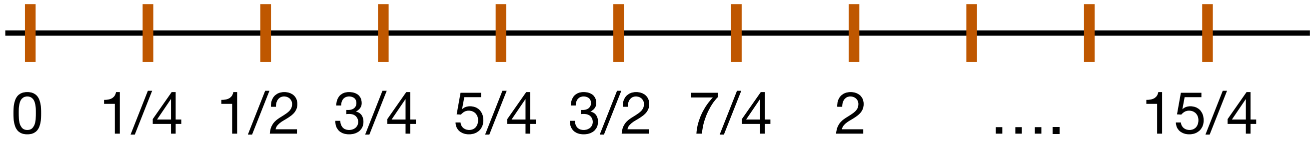
Limitations of Fixed-Point (#*1*)

Limitations of Fixed-Point (#*1*)

- Can exactly represent numbers only of the form $x/2^k$

Limitations of Fixed-Point (#*1*)

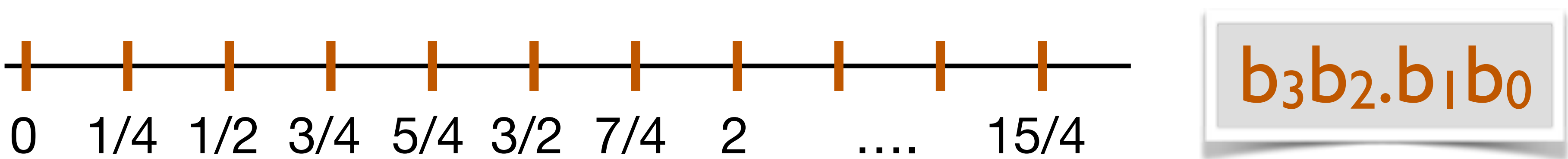
- Can exactly represent numbers only of the form $x/2^k$



$b_3b_2.b_1b_0$

Limitations of Fixed-Point (#*1*)

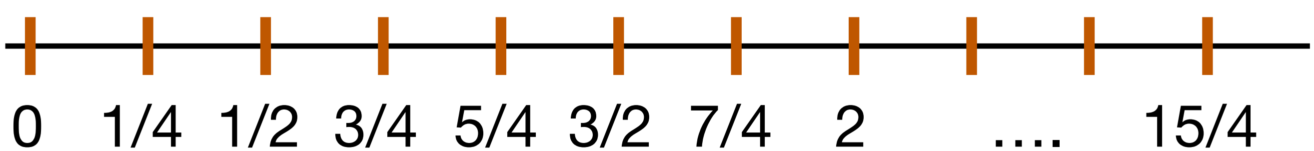
- Can exactly represent numbers only of the form $x/2^k$
  - Other rational numbers have repeating bit representations



Limitations of Fixed-Point (#*1*)

- Can exactly represent numbers only of the form $x/2^k$
  - Other rational numbers have repeating bit representations

Decimal Value	**Binary Representation**
1/3	0.0101010101[01]…
1/5	0.001100110011[0011]…
1/10	0.0001100110011[0011]…



$b_3b_2.b_1b_0$

Limitations of Fixed-Point (#2)

Limitations of Fixed-Point (#2)

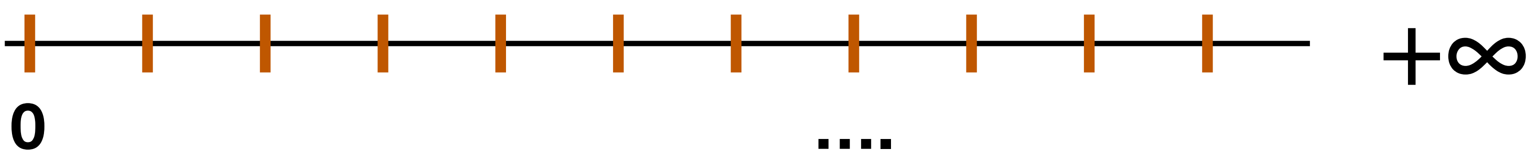
- Can’t represent very small and very large numbers at the same time

Limitations of Fixed-Point (#2)

- Can't represent very small and very large numbers at the same time
  - To represent very large numbers, the (fixed) interval needs to be large, making it hard to represent small numbers

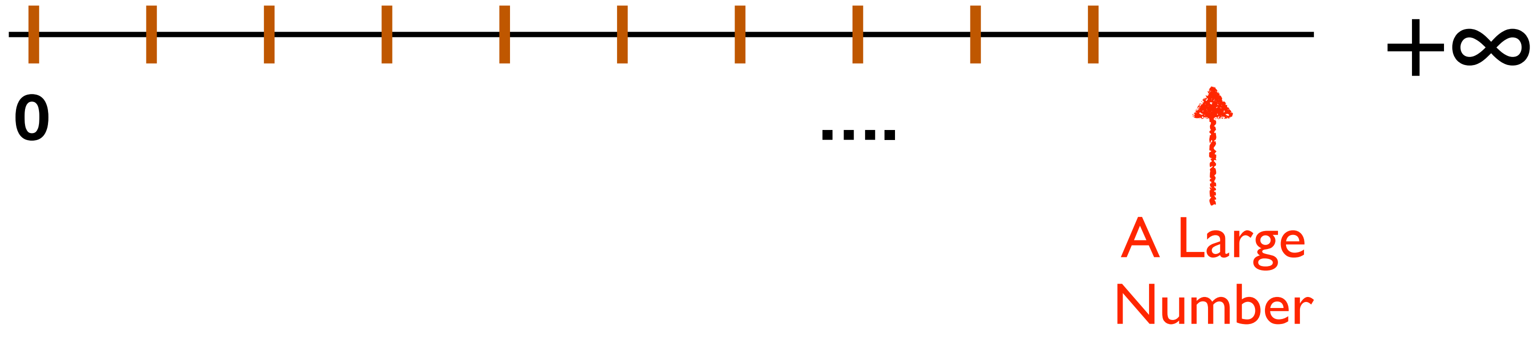
Limitations of Fixed-Point (#2)

- Can't represent very small and very large numbers at the same time
  - To represent very large numbers, the (fixed) interval needs to be large, making it hard to represent small numbers



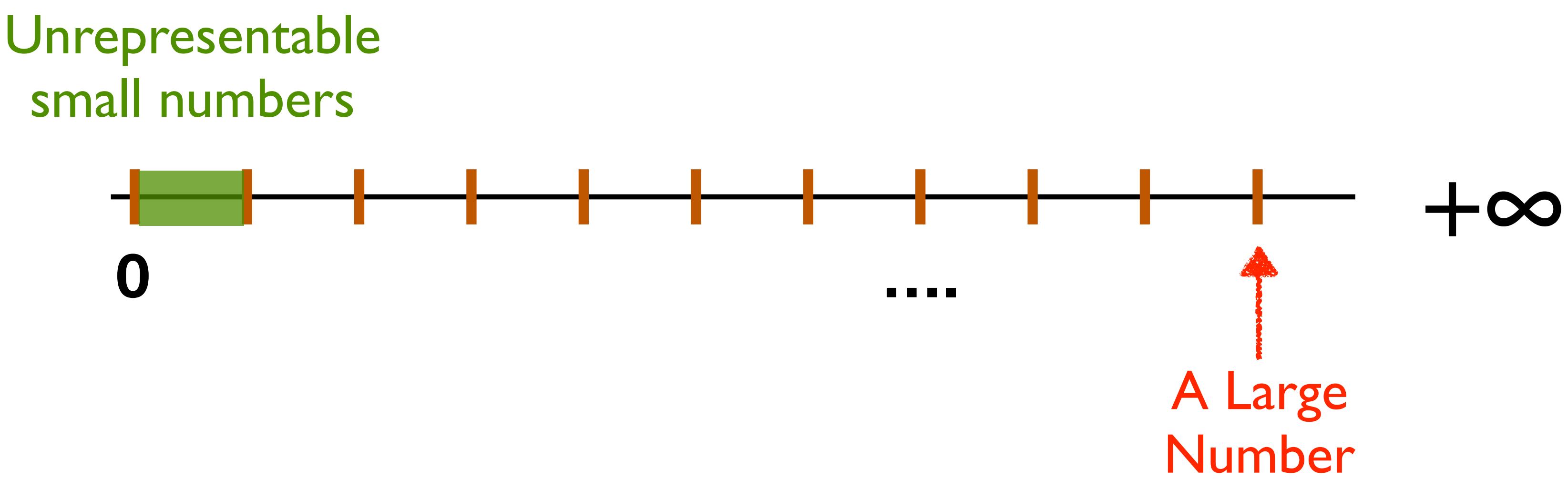
Limitations of Fixed-Point (#2)

- Can't represent very small and very large numbers at the same time
  - To represent very large numbers, the (fixed) interval needs to be large, making it hard to represent small numbers



Limitations of Fixed-Point (#2)

- Can't represent very small and very large numbers at the same time
  - To represent very large numbers, the (fixed) interval needs to be large, making it hard to represent small numbers



Limitations of Fixed-Point (#2)

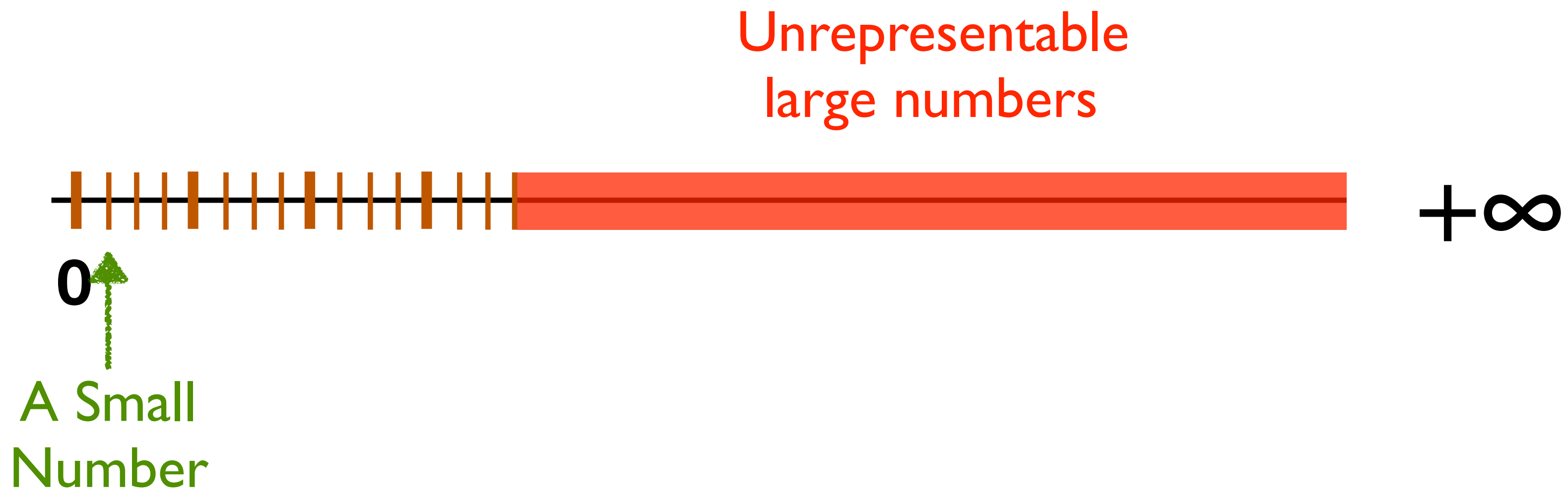
- Can't represent very small and very large numbers at the same time
  - To represent very large numbers, the (fixed) interval needs to be large, making it hard to represent small numbers
  - To represent very small numbers, the (fixed) interval needs to be small, making it hard to represent large numbers

Limitations of Fixed-Point (#2)

- Can't represent very small and very large numbers at the same time
  - To represent very large numbers, the (fixed) interval needs to be large, making it hard to represent small numbers
  - To represent very small numbers, the (fixed) interval needs to be small, making it hard to represent large numbers

Limitations of Fixed-Point (#2)

- Can't represent very small and very large numbers at the same time
  - To represent very large numbers, the (fixed) interval needs to be large, making it hard to represent small numbers
  - To represent very small numbers, the (fixed) interval needs to be small, making it hard to represent large numbers



Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

Primer: (Normalized) Scientific Notation

- In decimal: $M \times 10^E$
  - *E* is an integer
  - Normalized form: $1 <= |M| < 10$

Primer: (Normalized) Scientific Notation

- In decimal: $M \times 10^E$
  - *E* is an integer
  - Normalized form: $1 <= |M| < 10$

Decimal Value	Scientific Notation
2	2×10^0
-4,321.768	-4.321768×10^3
0.000 000 007 51	7.51×10^{-9}

Primer: (Normalized) Scientific Notation

- In decimal: $M \times 10^E$
  - *E* is an integer
  - Normalized form: $1 <= |M| < 10$

$$M \times 10^E$$

Decimal Value	Scientific Notation
2	2×10^0
-4,321.768	-4.321768×10^3
0.000 000 007 51	7.51×10^{-9}

Primer: (Normalized) Scientific Notation

- In decimal: $M \times 10^E$
  - E is an integer
  - Normalized form: $1 <= |M| < 10$

$$M \times 10^E$$

Significand

Decimal Value	Scientific Notation
2	2×10^0
-4,321.768	-4.321768×10^3
0.000 000 007 51	7.51×10^{-9}

Primer: (Normalized) Scientific Notation

- In decimal: $M \times 10^E$
  - E is an integer
  - Normalized form: $1 <= |M| < 10$

$$\underset{\text{Significand}}{\underset{\uparrow}{M}} \times \underset{\text{Base}}{\underset{\uparrow}{10}}^{E}$$

Decimal Value	**Scientific Notation**
2	2×10^0
-4,321.768	-4.321768×10^3
0.000 000 007 51	7.51×10^{-9}

Primer: (Normalized) Scientific Notation

- In decimal: $M \times 10^E$
  - E is an integer
  - Normalized form: $1 <= |M| < 10$

$$M \times 10^{E}$$

Exponent

Significand

Base

Decimal Value	Scientific Notation
2	2×10^0
-4,321.768	-4.321768×10^3
0.000 000 007 51	7.51×10^{-9}

Primer: (Normalized) Scientific Notation

- In binary: $(-1)^s M\ 2^E$
- Normalized form:
  - $1 <= M < 10$
  - $M = 1.b_0b_1b_2b_3\ldots$

$$(-1)^s M \times 2^E$$

Primer: (Normalized) Scientific Notation

- In binary: $(-1)^s \, M \, 2^E$
- Normalized form:
  - $1 <= M < 10$
  - $M = 1.b_0 b_1 b_2 b_3 \ldots$

$$(-1)^s \, M \times 2^E$$

Base

Primer: (Normalized) Scientific Notation

- In binary: $(-1)^s \, M \, 2^E$
- Normalized form:
  - $1 <= M < 10$
  - $M = 1.b_0b_1b_2b_3\ldots$

Exponent

$$(-1)^s \, M \times 2^E$$

Base

Primer: (Normalized) Scientific Notation

- In binary: $(-1)^s\ M\ 2^E$
- Normalized form:
  - $1 <= M < 10$
  - $M = 1.b_0b_1b_2b_3\ldots$

$$(-1)^{s}\ \underbrace{M}_{\text{Significand}} \times \underbrace{2}_{\text{Base}}^{\overset{\text{Exponent}}{E}}$$

Primer: (Normalized) Scientific Notation

- In binary: $(-1)^s\, M\, 2^E$
- Normalized form:
  - $1 <= M < 10$
  - $M = 1.b_0b_1b_2b_3\ldots$

$$(-1)^{\overset{\text{Sign}}{\downarrow}{s}}\ \underset{\underset{\text{Significand}}{\uparrow}}{M} \times \underset{\underset{\text{Base}}{\uparrow}}{2}^{\overset{\text{Exponent}}{\downarrow}{E}}$$

Primer: (Normalized) Scientific Notation

- In binary: $(-1)^s\, M\, 2^E$
- Normalized form:
  - $1 <= M < 10$
  - $M = 1.b_0b_1b_2b_3\ldots$

    Fraction

Sign Exponent

$$(-1)^{s}\, M \times 2^{E}$$

Significand Base

Primer: (Normalized) Scientific Notation

- In binary: $(-1)^{s}\ M\ 2^{E}$
- Normalized form:
  - $1 <= M < 10$
  - $M = 1.\underbrace{b_0 b_1 b_2 b_3 \ldots}_{\text{Fraction}}$

$$(-1)^{\overset{\text{Sign}}{\downarrow} s}\ \underset{\text{Significand}}{\underset{\uparrow}{M}} \times \underset{\text{Base}}{\underset{\uparrow}{2}}^{\overset{\text{Exponent}}{\downarrow} E}$$

Binary Value	Scientific Notation
1110110110110	$(-1)^{0}$ 1.110110110110 x 2^{12}
-101.11	$(-1)^{1}$ 1.0111 x 2^{2}
0.00101	$(-1)^{0}$ 1.01 x 2^{-3}

Primer: Floating Point Representation

- In binary: $(-1)^s\ M\ 2^E$
- Normalized form:
  - $1 <= M < 2$
  - $M = 1.\underbrace{b_0 b_1 b_2 b_3 \ldots}_{\text{Fraction}}$

Sign Exponent

$$(-1)^s\ M \times 2^E$$

Significand Base

Primer: Floating Point Representation

- In binary: $(-1)^{s}\, M\, 2^{E}$
- Normalized form:
  - $1 <= M < 2$
  - $M = 1.\underbrace{b_0 b_1 b_2 b_3 \ldots}_{\text{Fraction}}$
- Encoding

$$(-1)^{\overset{\text{Sign}}{\downarrow}{s}}\ \underset{\overset{\uparrow}{\text{Significand}}}{M} \times \underset{\overset{\uparrow}{\text{Base}}}{2}^{\overset{\text{Exponent}}{\downarrow}{E}}$$

Primer: Floating Point Representation

- In binary: $(-1)^s\ M\ 2^E$
- Normalized form:
  - $1 <= M < 2$
  - $M = 1.b_0b_1b_2b_3\ldots$

    Fraction
- Encoding

Sign Exponent

$$(-1)^{s}\ M \times 2^{E}$$

Significand Base

Primer: Floating Point Representation

- In binary: $(-1)^s M\, 2^E$
- Normalized form:
  - $1 <= M < 2$
  - $M = 1.b_0b_1b_2b_3\ldots$

    Fraction
- Encoding
  - MSB s is sign bit s

Sign Exponent

$$(-1)^{s}\, M \times 2^{E}$$

Significand Base

Primer: Floating Point Representation

- In binary: $(-1)^s\, M\, 2^E$
- Normalized form:
  - $1 <= M < 2$
  - $M = 1.b_0 b_1 b_2 b_3 \ldots$

    Fraction
- Encoding
  - MSB *s* is sign bit s
  - *exp* field encodes Exponent (but not exactly the same as E)

Sign Exponent

$$(-1)^s\, M \times 2^E$$

Significand Base

Primer: Floating Point Representation

- In binary: $(-1)^s M 2^E$
- Normalized form:
  - $1 <= M < 2$
  - $M = 1.\underbrace{b_0 b_1 b_2 b_3 \ldots}_{\text{Fraction}}$

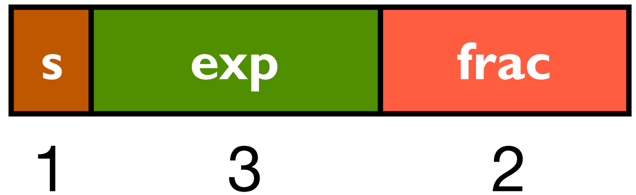
$$\overset{\text{Sign}}{(-1)^{s}}\ \underset{\text{Significand}}{M} \times \underset{\text{Base}}{2}^{\overset{\text{Exponent}}{E}}$$

- Encoding
  - MSB *s* is sign bit s
  - *exp* field encodes Exponent (but not exactly the same as E)
  - *frac* field encodes Fraction (but not exactly the same as Fraction)

s	**exp**	**frac**

6-bit Floating Point Example

$v = (-1)^s M\, 2^E$



6-bit Floating Point Example

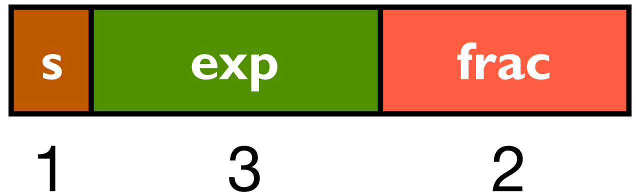
$$v = (-1)^s M 2^E$$

s	exp	frac
1	3	2

- *exp* has 3 bits, interpreted as an unsigned value

6-bit Floating Point Example

$$v = (-1)^s \, M \, 2^E$$



- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**

6-bit Floating Point Example

$$v = (-1)^s \, M \, 2^E$$

s	exp	frac
1	3	2

- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?

6-bit Floating Point Example

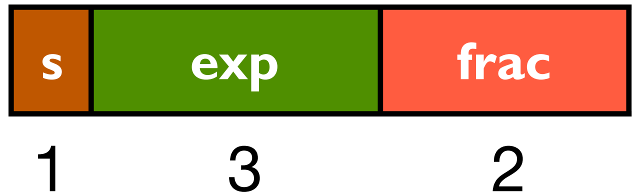
$$v = (-1)^s \, M \, 2^E$$

s	exp	frac
1	3	2

- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?
  - Add a bias term: *E = exp - bias* (i.e., exp = E + bias)

6-bit Floating Point Example

$$v = (-1)^s \, M \, 2^E$$



- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?
  - Add a bias term: *E = exp - bias* (i.e., exp = E + bias)
  - bias is always $2^{k-1} - 1$, where k is number of exponent bits

6-bit Floating Point Example

$$v = (-1)^s \, M \, 2^E$$

s	exp	frac
1	3	2

- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?
  - Add a bias term: *E = exp - bias* (i.e., exp = E + bias)
  - bias is always $2^{k-1} - 1$, where k is number of exponent bits
- Example when *k* = 3:

6-bit Floating Point Example

$$v = (-1)^s \, M \, 2^E$$

s	exp	frac
1	3	2

- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?
  - Add a bias term: *E = exp - bias* (i.e., exp = E + bias)
  - bias is always $2^{k-1} - 1$, where k is number of exponent bits
- Example when *k* = 3:
  - bias = 3

6-bit Floating Point Example

$v = (-1)^s M 2^E$

s	exp	frac
1	3	2

- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?
  - Add a bias term: *E = exp - bias* (i.e., exp = E + bias)
  - bias is always $2^{k-1} - 1$, where k is number of exponent bits
- Example when *k* = 3:
  - bias = 3
  - If *E* = -2, *exp* is 1 (001_2)

6-bit Floating Point Example

$v = (-1)^s M\, 2^E$

s	exp	frac
1	3	2

- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?
  - Add a bias term: *E = exp - bias* (i.e., exp = E + bias)
  - bias is always $2^{k-1} - 1$, where k is number of exponent bits
- Example when $k = 3$:
  - bias = 3
  - If $E = -2$, *exp* is 1 (001_2)

E	exp
-3	000
-2	001
-1	010
0	011
1	100
2	101
3	110
4	111

6-bit Floating Point Example

$$v = (-1)^s \, M \, 2^E$$

s	exp	frac
1	3	2

- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?
  - Add a bias term: *E = exp - bias* (i.e., exp = E + bias)
  - bias is always $2^{k-1} - 1$, where k is number of exponent bits
- Example when $k = 3$:
  - bias = 3
  - If $E = -2$, *exp* is 1 (001_2)
  - Reserve 000 and 111 for other purposes (more on this later)

E	exp
~~-3~~	~~000~~
-2	001
-1	010
0	011
1	100
2	101
3	110
~~4~~	~~111~~

6-bit Floating Point Example

$$v = (-1)^s M\, 2^E$$

s	exp	frac
1	3	2

- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?
  - Add a bias term: *E = exp - bias* (i.e., exp = E + bias)
  - bias is always 2^{k-1} - 1, where k is number of exponent bits
- Example when *k* = 3:
  - bias = 3
  - If *E* = -2, *exp* is 1 (001_2)
  - Reserve 000 and 111 for other purposes (more on this later)
  - We can now represent exponents from **-2 (exp 001) to 3 (exp 110)**

E	exp
~~-3~~	~~000~~
-2	001
-1	010
0	011
1	100
2	101
3	110
~~4~~	~~111~~

6-bit Floating Point Example

$v = (-1)^s \, M \, 2^E$

s	exp	frac
1	3	2

6-bit Floating Point Example

$v = (-1)^s M 2^E$

- *frac* has 2 bits, append them after “1.” to form M
  - *frac* = 10 implies M = 1.10

6-bit Floating Point Example

$$v = (-1)^s M 2^E$$

s	exp	frac
1	3	2

- *frac* has 2 bits, append them after "1." to form M
  - *frac* = 10 implies M = 1.10
- Putting it Together: An Example:

$$-10.1_2 = (-1)^1\ 1.01 \times 2^1$$

6-bit Floating Point Example

$v = (-1)^s\, M\, 2^E$

s	exp	frac
1	3	2

- *frac* has 2 bits, append them after "1." to form M
  - *frac* = 10 implies M = 1.10
- Putting it Together: An Example:

$$-10.1_2 = (-1)^1\ 1.01 \times 2^1$$

6-bit Floating Point Example

$v = (-1)^s \, M \, 2^E$

1	exp	frac
1	3	2

- *frac* has 2 bits, append them after "1." to form M
  - *frac* = 10 implies M = 1.10
- Putting it Together: An Example:

$$-10.1_2 = (-1)^1 \; 1.01 \times 2^1$$

6-bit Floating Point Example

$v = (-1)^s \, M \, 2^E$

1	exp	frac
1	3	2

- *frac* has 2 bits, append them after “1.” to form M
  - *frac* = 10 implies M = 1.10
- Putting it Together: An Example:

$$-10.1_2 = (-1)^1 \, 1.01 \times 2^1$$

6-bit Floating Point Example

$v = (-1)^s M\, 2^E$

1	exp	frac
1	3	2

- *frac* has 2 bits, append them after “1.” to form M
  - *frac* = 10 implies M = 1.10
- Putting it Together: An Example:

$$-10.1_2 = (-1)^1\ 1.01 \times 2^1$$

E	exp
~~-3~~	~~000~~
-2	001
-1	010
0	011
1	100
2	101
3	110
~~4~~	~~111~~

6-bit Floating Point Example

$v = (-1)^s M\, 2^E$

1	100	frac
1	3	2

- *frac* has 2 bits, append them after “1.” to form M
  - *frac* = 10 implies M = 1.10
- Putting it Together: An Example:

$$-10.1_2 = (-1)^1\ 1.01 \times 2^1$$

E	exp
~~-3~~	~~000~~
-2	001
-1	010
0	011
1	100
2	101
3	110
~~4~~	~~111~~

6-bit Floating Point Example

$v = (-1)^s M 2^E$

1	100	frac
1	3	2

- *frac* has 2 bits, append them after “1.” to form M
  - *frac* = 10 implies M = 1.10
- Putting it Together: An Example:

$$-10.1_2 = (-1)^1\ 1.01 \times 2^1$$

E	exp
~~-3~~	~~000~~
-2	001
-1	010
0	011
1	100
2	101
3	110
~~4~~	~~111~~

6-bit Floating Point Example

$$v = (-1)^s\, M\, 2^E$$

1	100	01
1	3	2

- *frac* has 2 bits, append them after "1." to form M
  - *frac* = 10 implies M = 1.10
- Putting it Together: An Example:

$$-10.1_2 = (-1)^1\ 1.01 \times 2^1$$

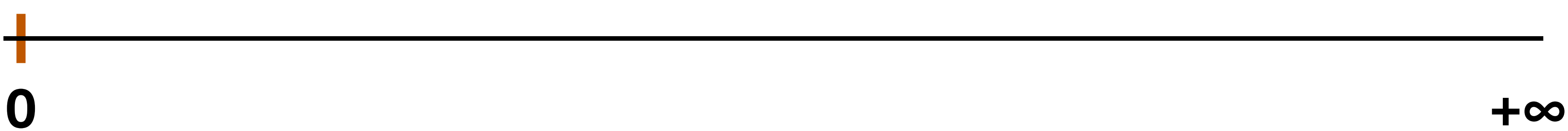
E	exp
~~-3~~	~~000~~
-2	001
-1	010
0	011
1	100
2	101
3	110
~~4~~	~~111~~

Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

s	exp	frac

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

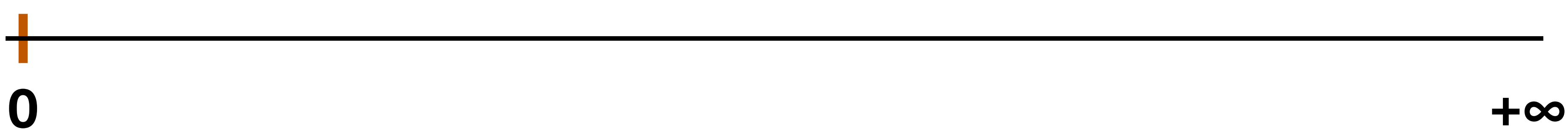


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	exp	frac

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

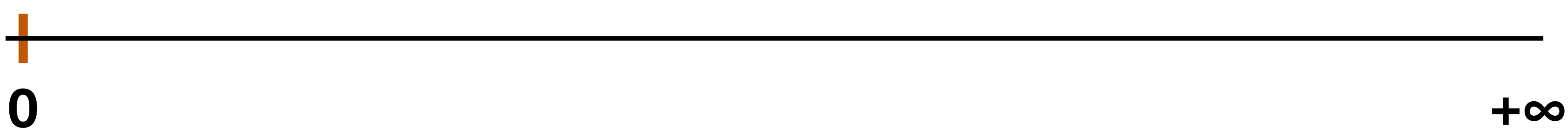


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	exp	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

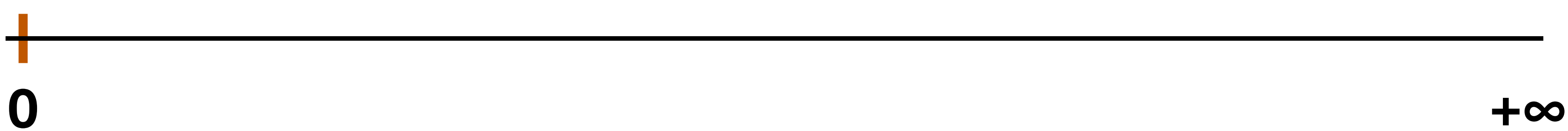


Representable Numbers (Positive Only)

$$v = (-1)^s \, M \, 2^E$$

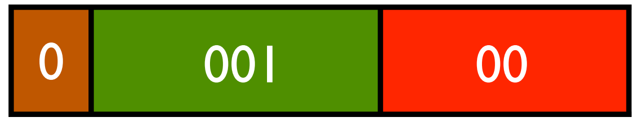
0	001	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~



Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$



E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

Representable Numbers (Positive Only)

$v = (-1)^s M\, 2^E$

0	010	00

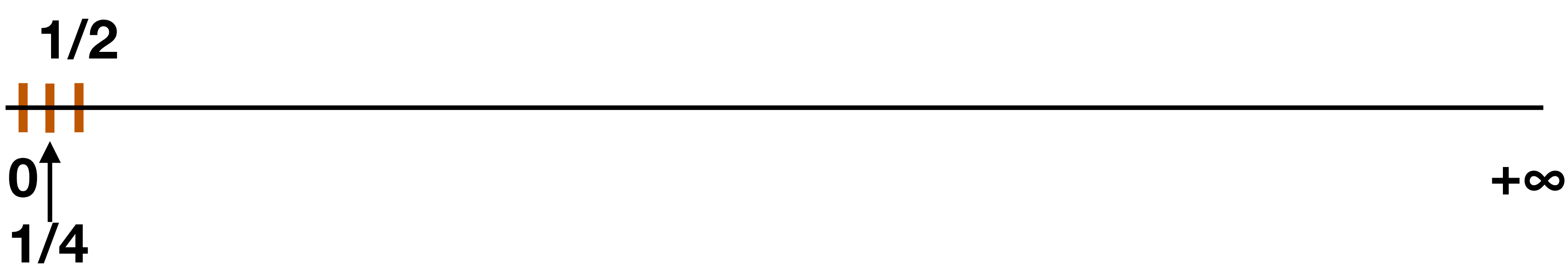
E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

Representable Numbers (Positive Only)

$v = (-1)^s\, M\, 2^E$

0	010	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

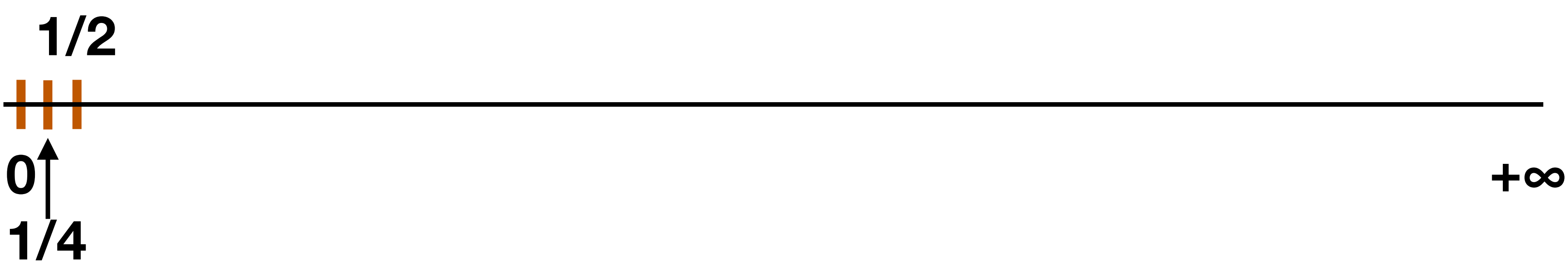


Representable Numbers (Positive Only)

$v = (-1)^s M\, 2^E$

0	011	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

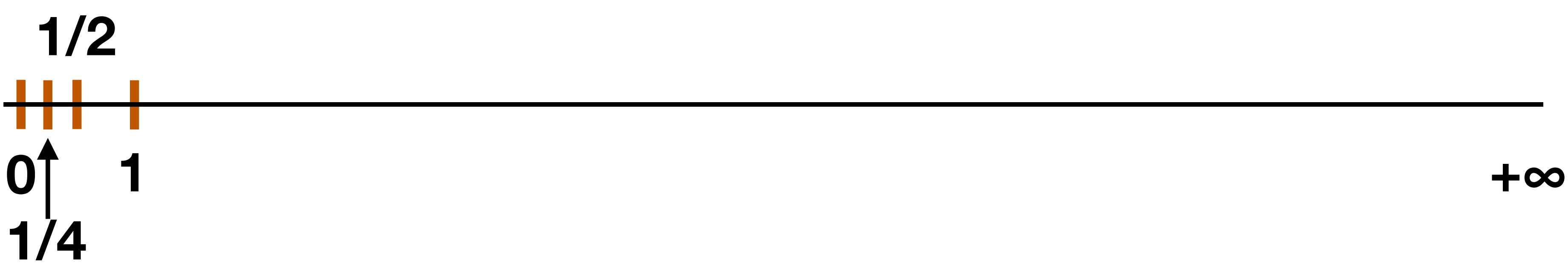


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	011	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

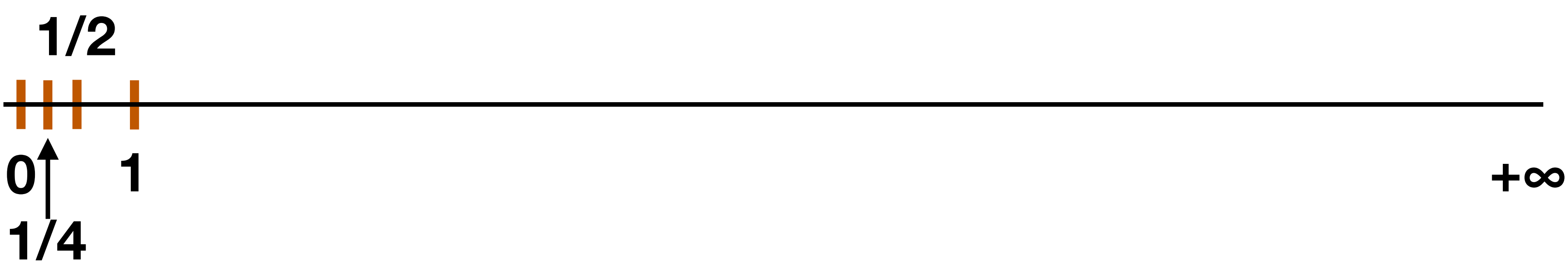


Representable Numbers (Positive Only)

$v = (-1)^s\, M\, 2^E$

0	100	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

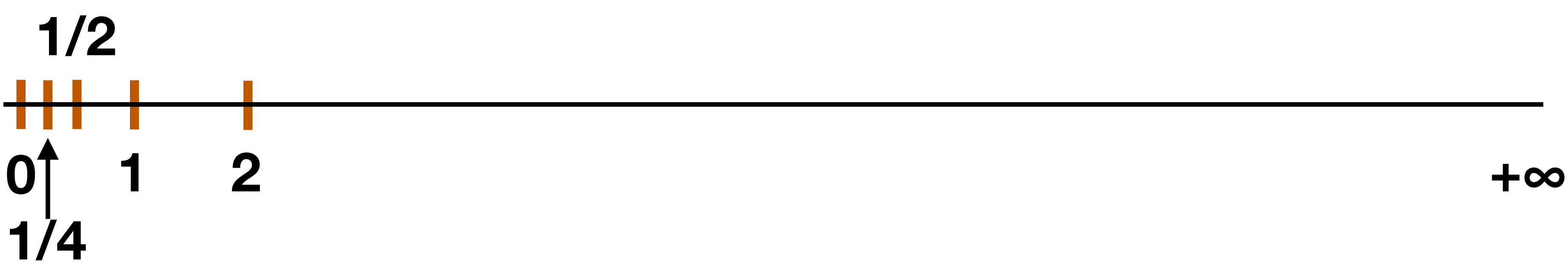


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	100	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

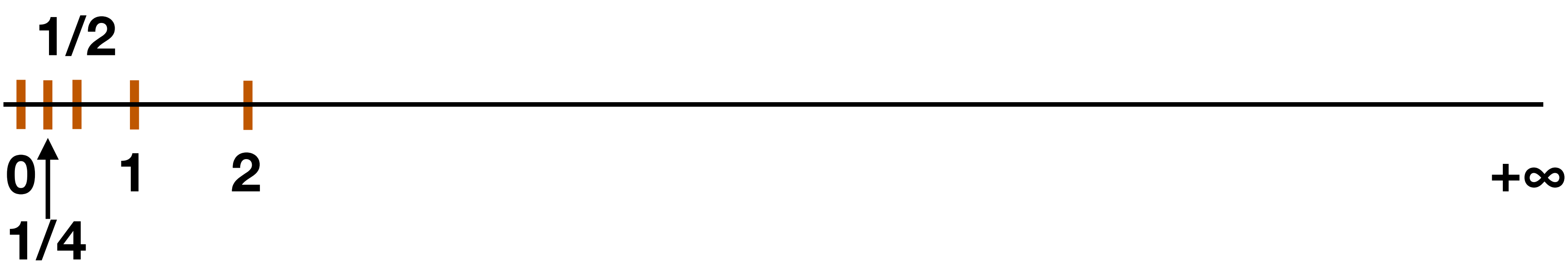


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	101	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

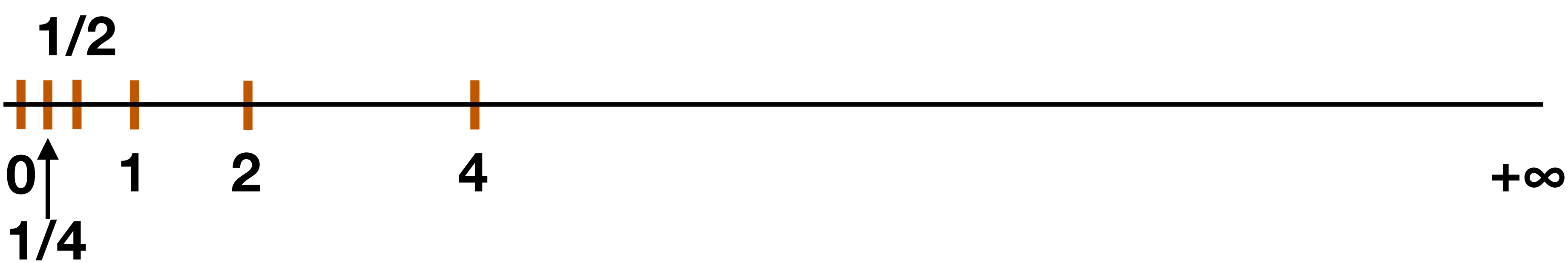


Representable Numbers (Positive Only)

$v = (-1)^s M\, 2^E$

0	101	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

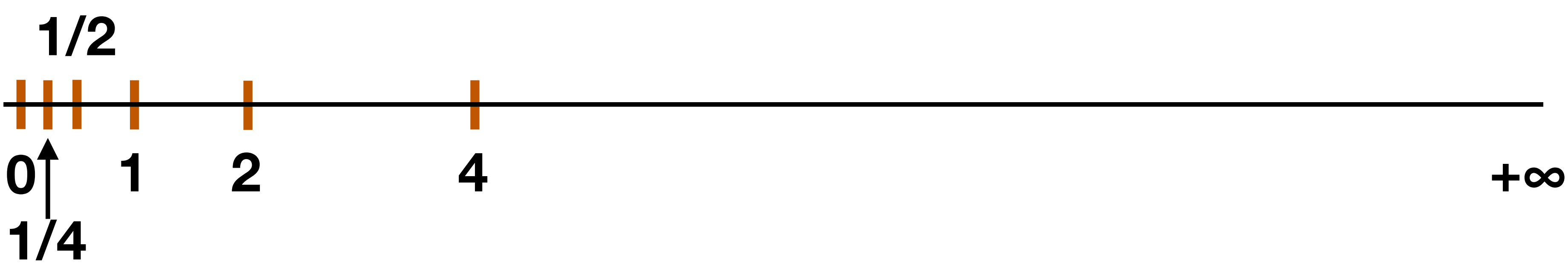


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	110	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

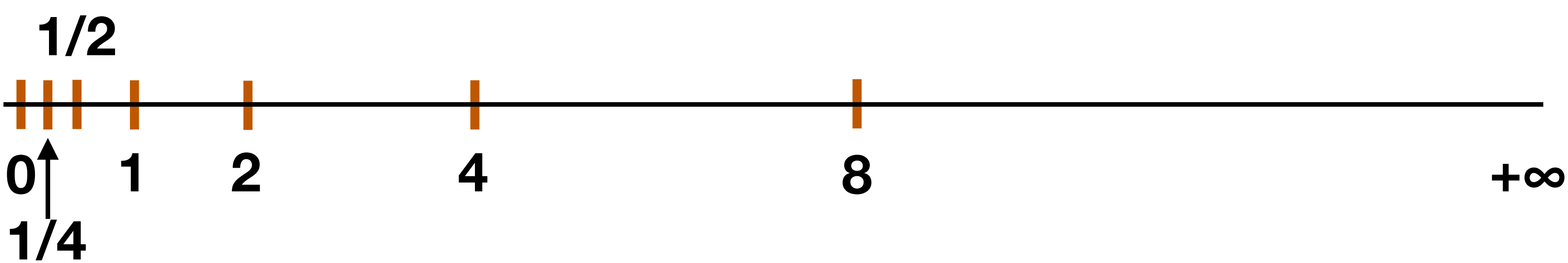


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	110	00

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

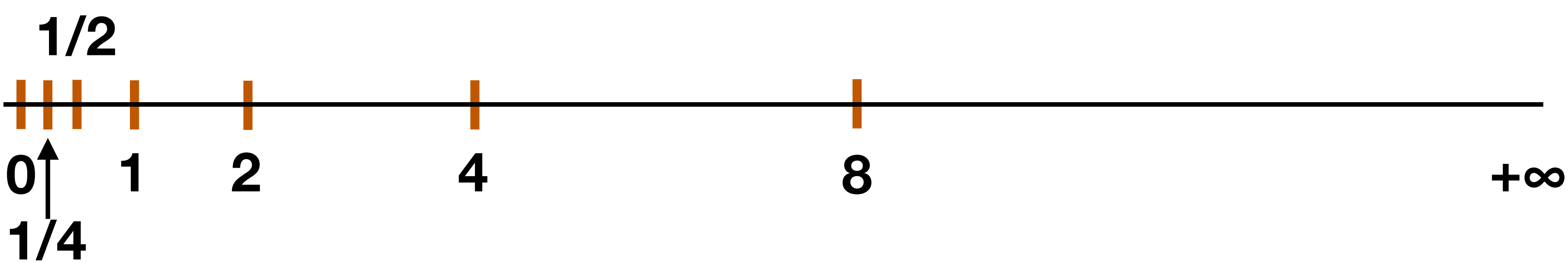


Representable Numbers (Positive Only)

$v = (-1)^s\, M\, 2^E$

0	110	01

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

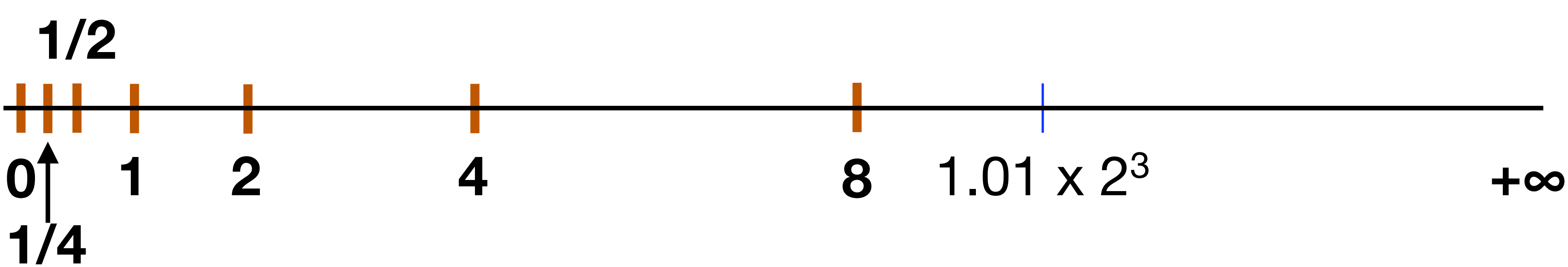


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	110	01

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

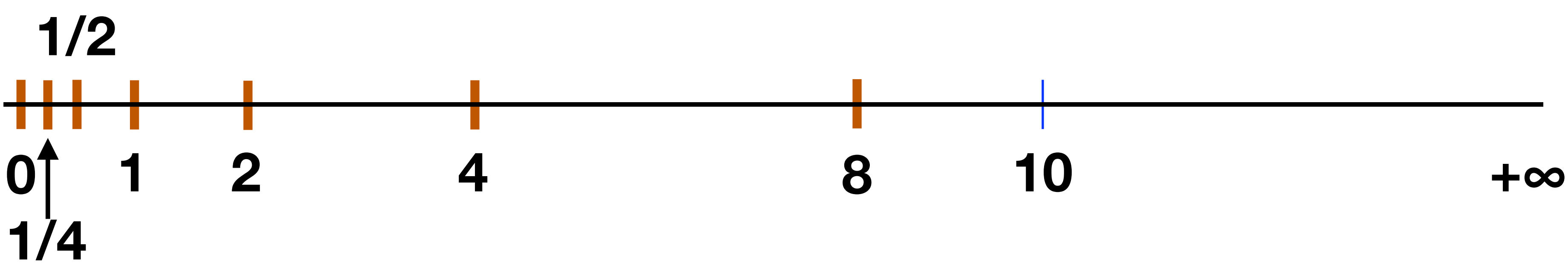


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	110	01

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

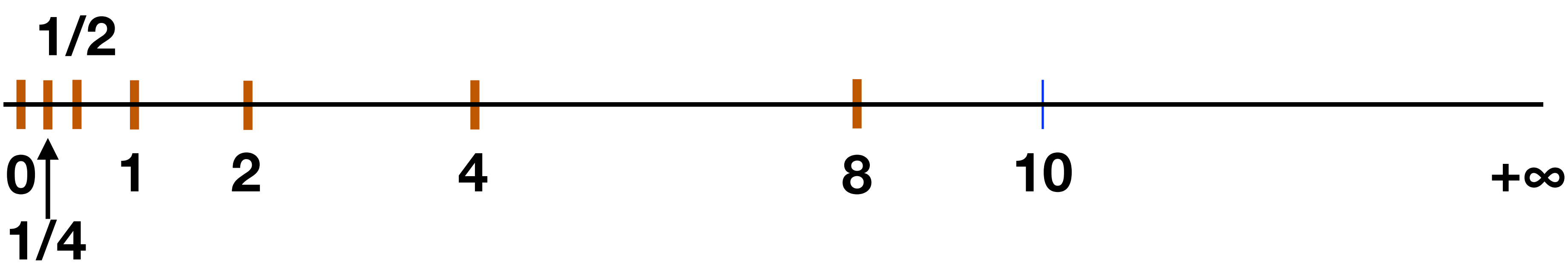


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	110	10

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

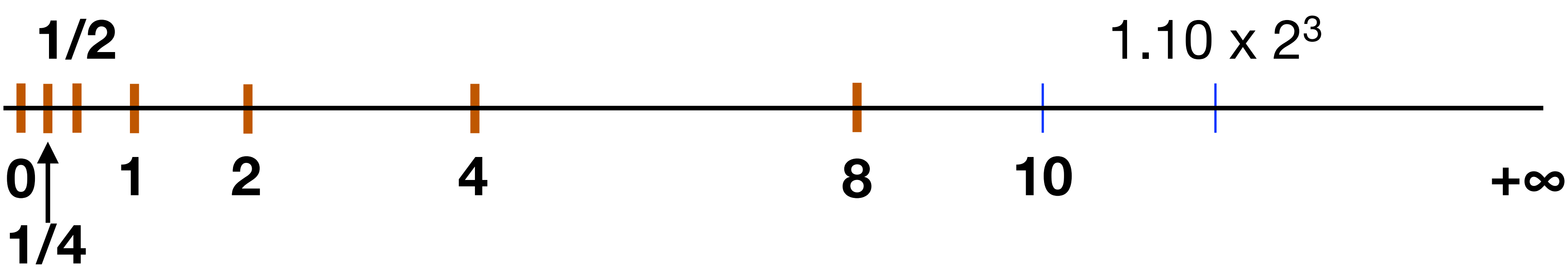


Representable Numbers (Positive Only)

$v = (-1)^s\, M\, 2^E$

0	110	10

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

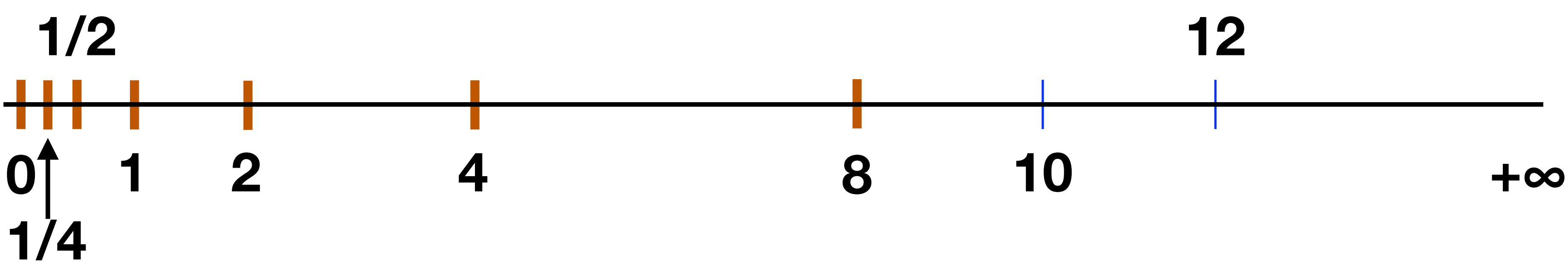


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	110	10

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

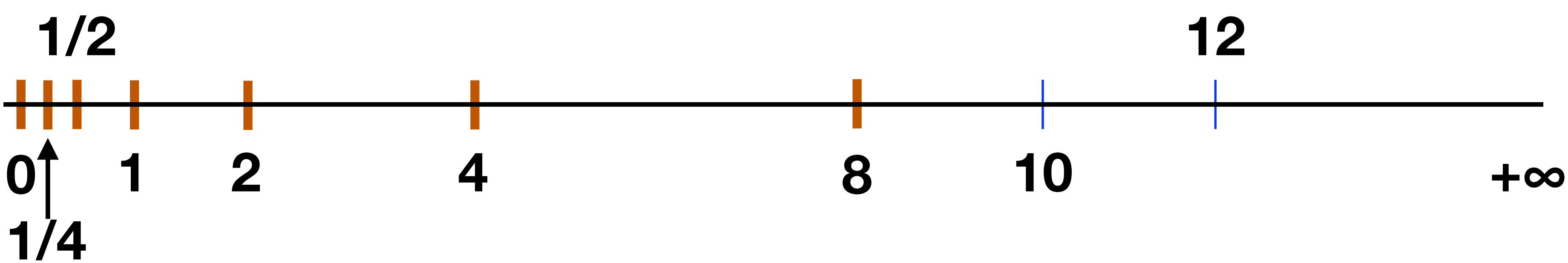


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	110	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

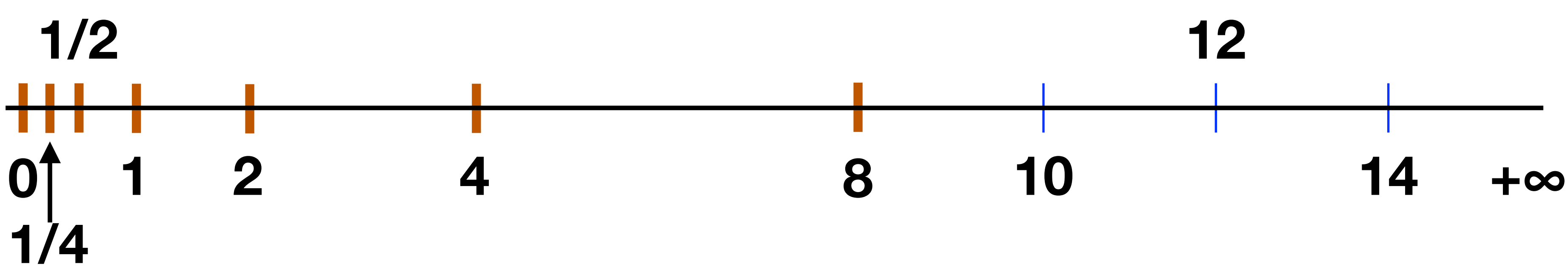


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	110	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

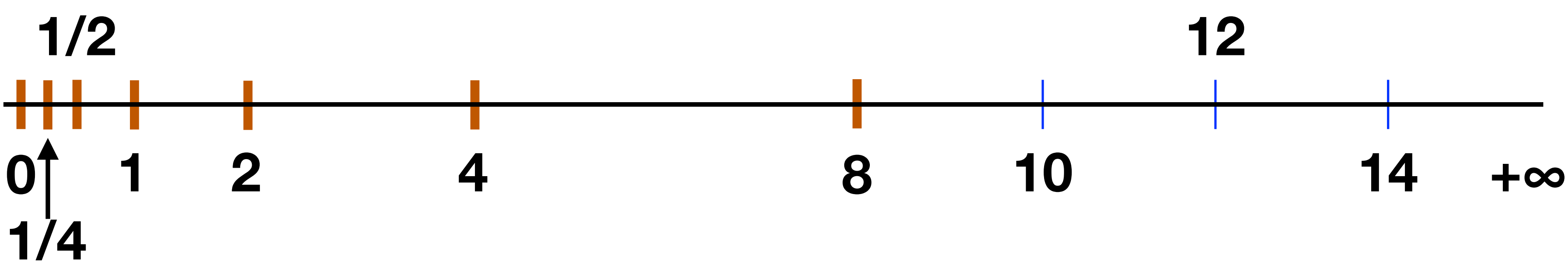


Representable Numbers (Positive Only)

$v = (-1)^s\ M\ 2^E$

0	101	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

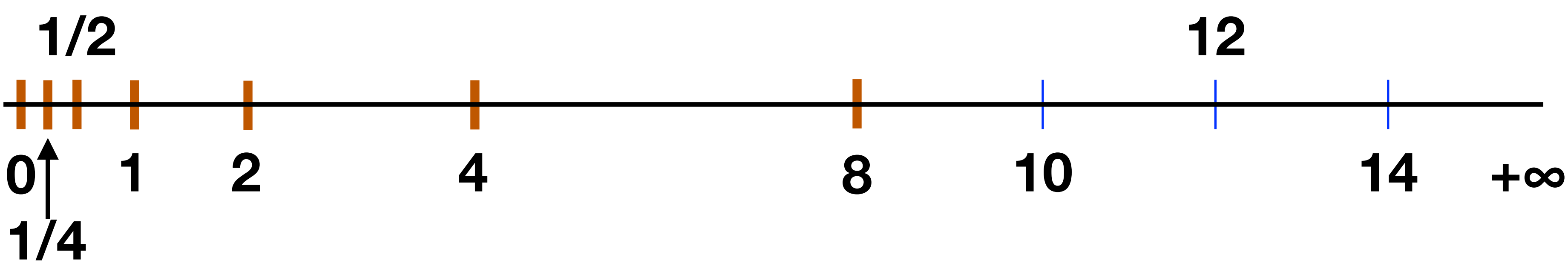


Representable Numbers (Positive Only)

$v = (-1)^s\, M\, 2^E$

0	101	01

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

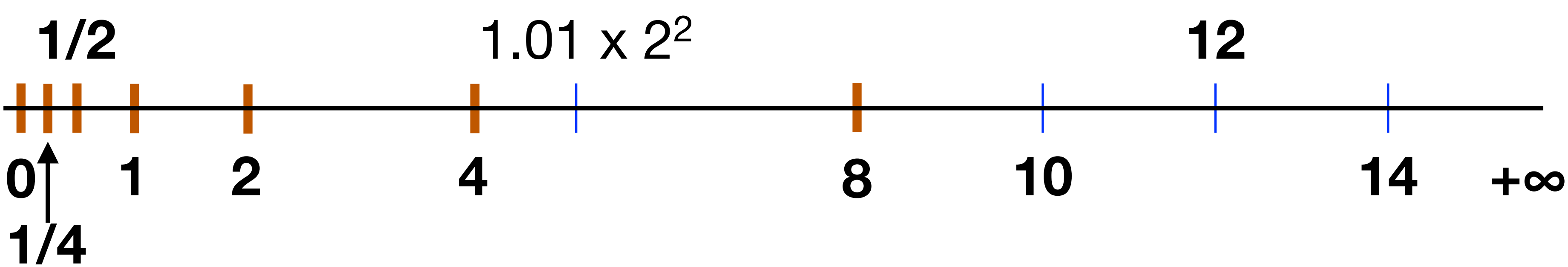


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

s	exp	frac
0	101	01

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

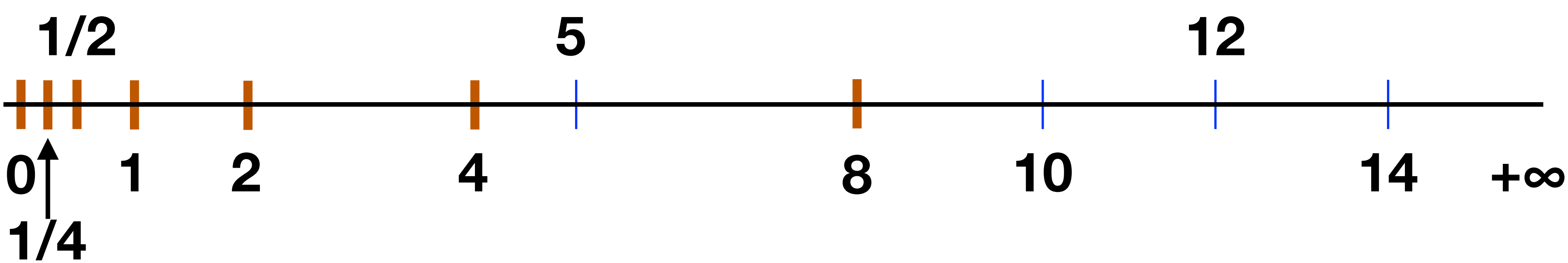


Representable Numbers (Positive Only)

$v = (-1)^s M\ 2^E$

0	101	01

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

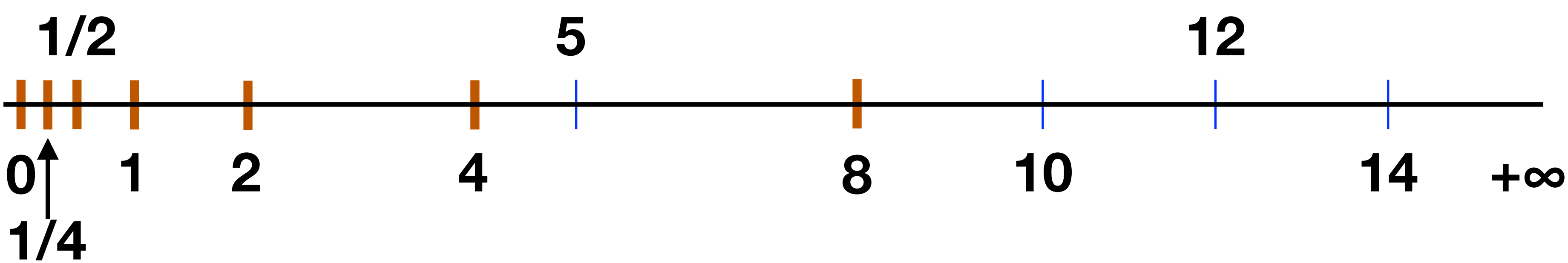


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	101	10

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

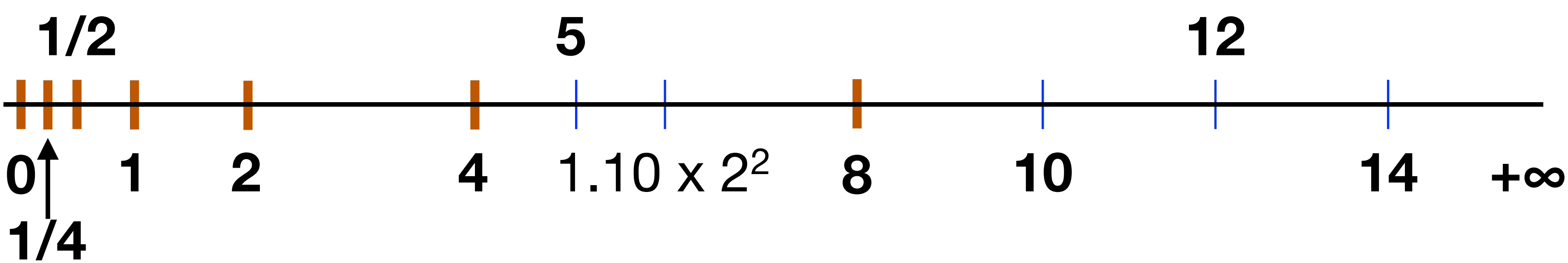


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	101	10

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

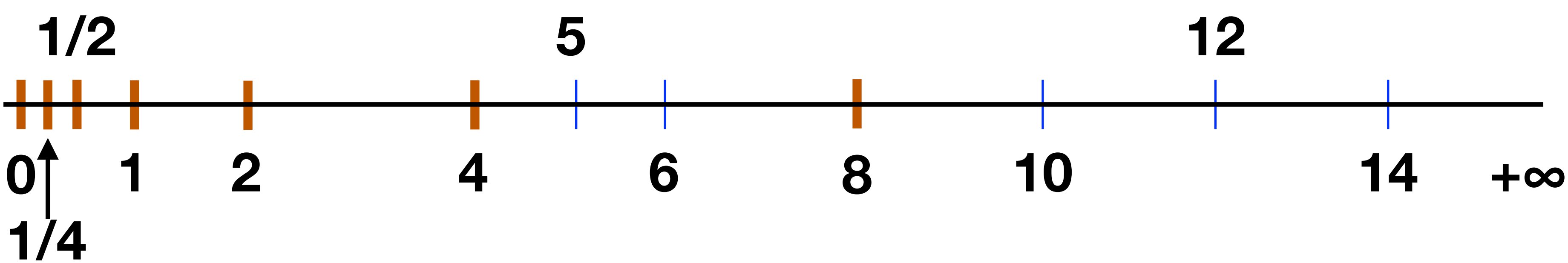


Representable Numbers (Positive Only)

$$v = (-1)^s M 2^E$$

0	101	10

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

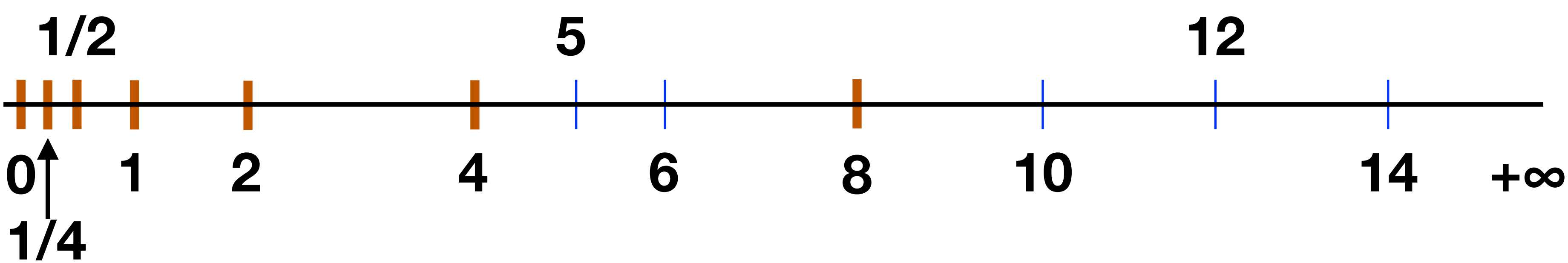


Representable Numbers (Positive Only)

$v = (-1)^s\, M\, 2^E$

0	101	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

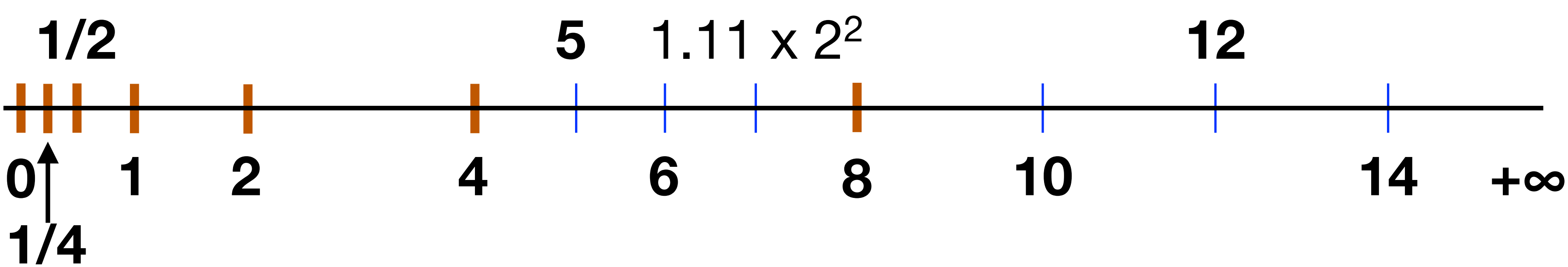


Representable Numbers (Positive Only)

$v = (-1)^s M \, 2^E$

0	101	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

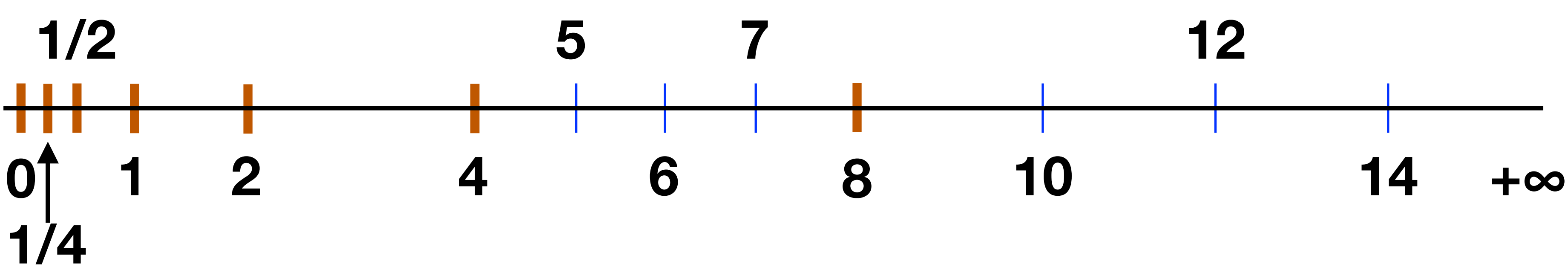


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	101	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

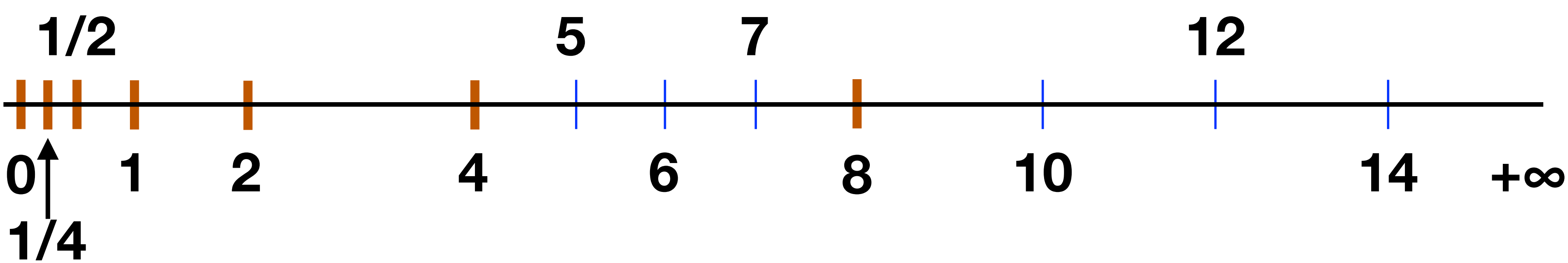


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	100	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

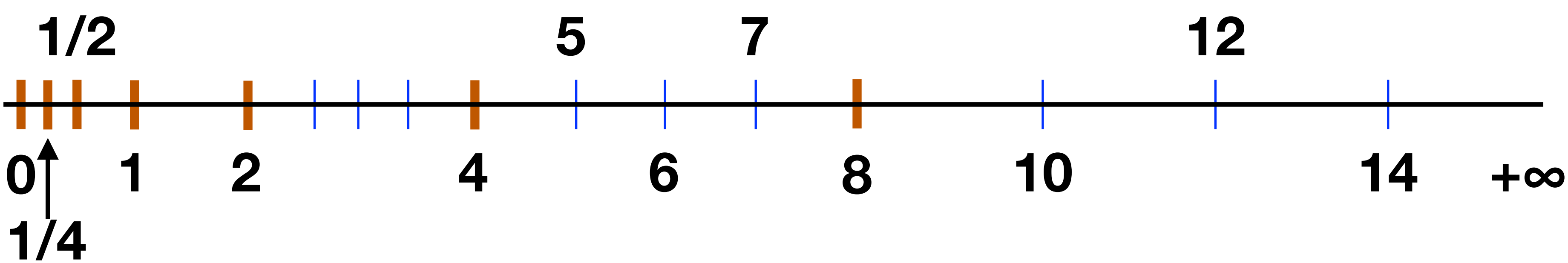


Representable Numbers (Positive Only)

$v = (-1)^s\, M\, 2^E$

0	100	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

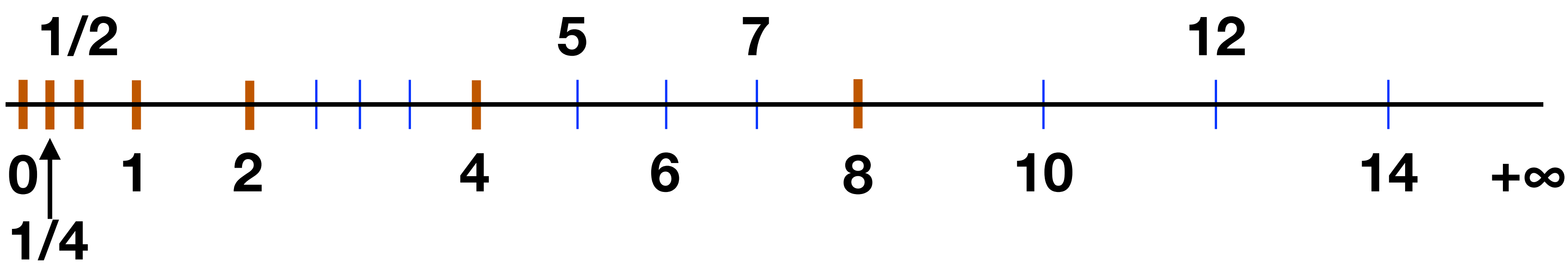


Representable Numbers (Positive Only)

$$v = (-1)^s \, M \, 2^E$$

0	011	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

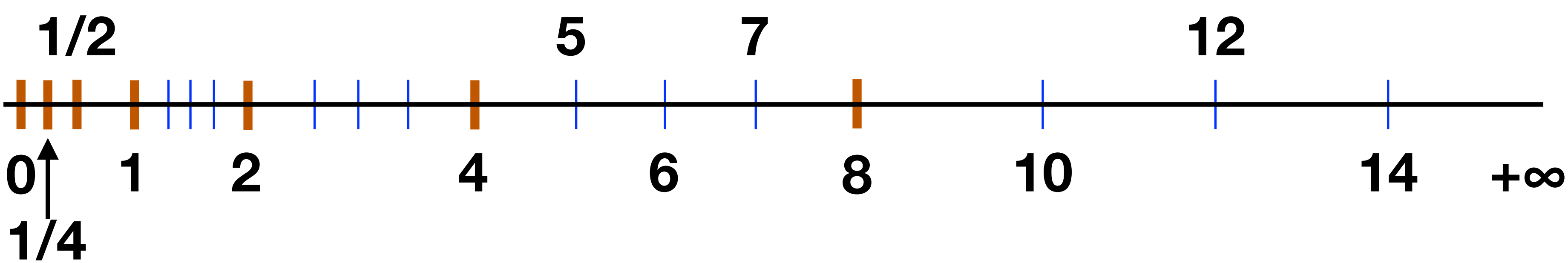


Representable Numbers (Positive Only)

$v = (-1)^s M\, 2^E$

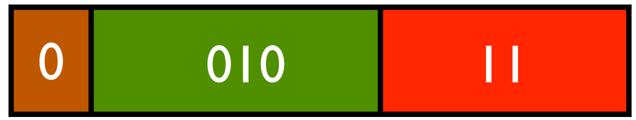
0	011	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

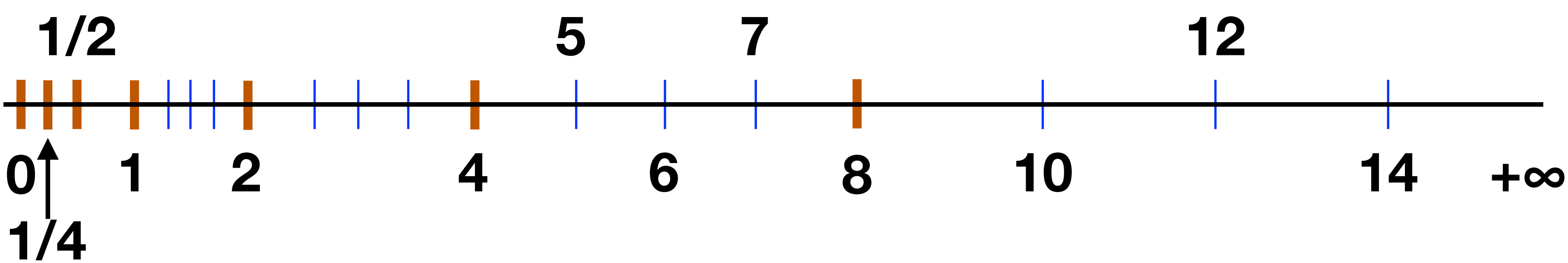


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$



E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

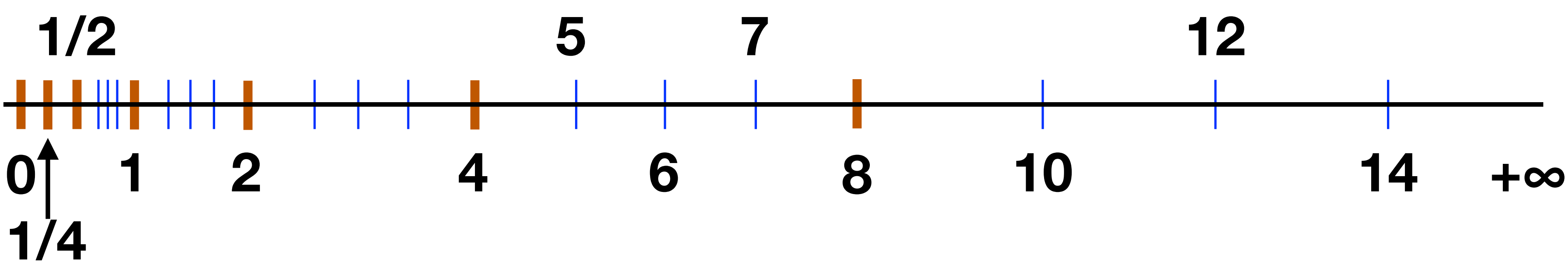


Representable Numbers (Positive Only)

$v = (-1)^s \, M \, 2^E$

0	010	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

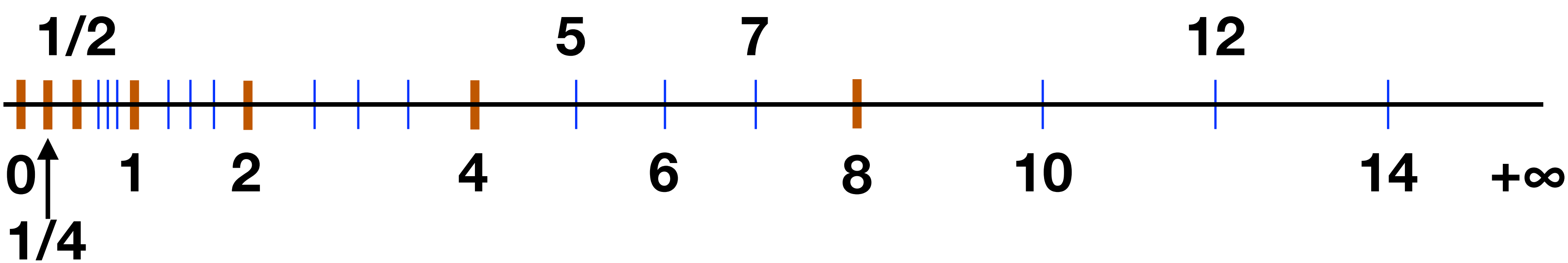


Representable Numbers (Positive Only)

$v = (-1)^s\, M\, 2^E$

0	001	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

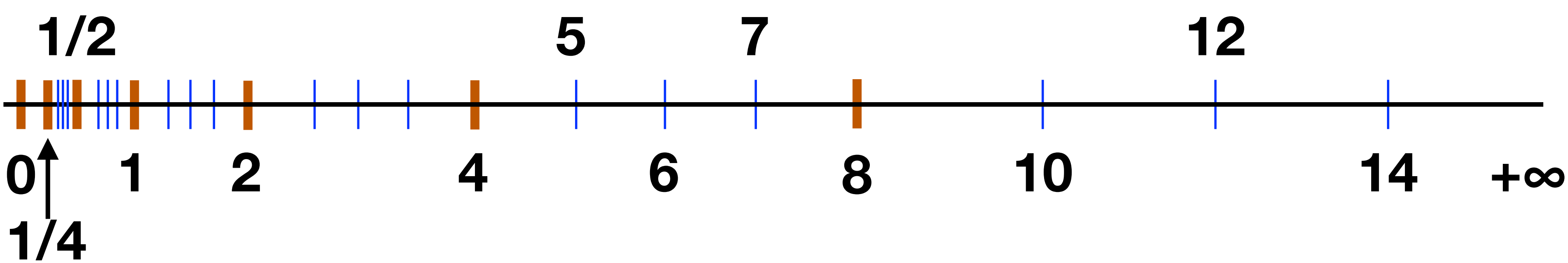


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	001	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~



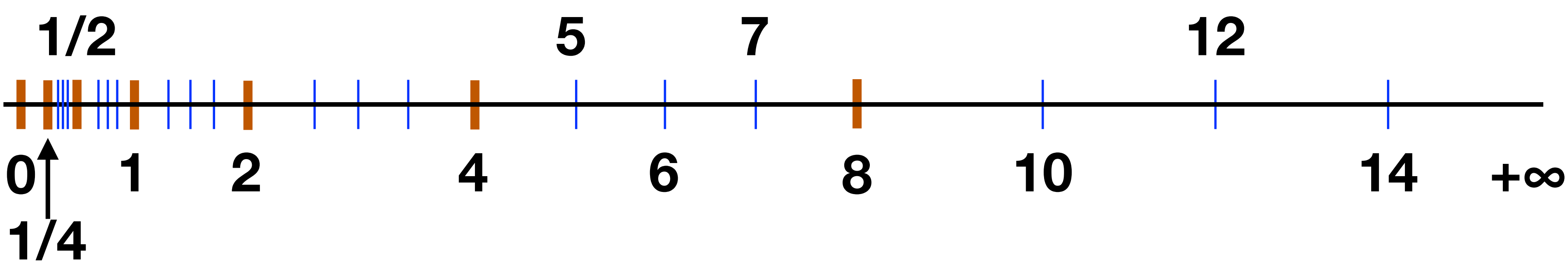
Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	001	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Uneven interval (c.f., fixed interval in fixed-point)
  - More dense toward 0, sparser toward infinite
  - Allow encoding small and large numbers at the same time



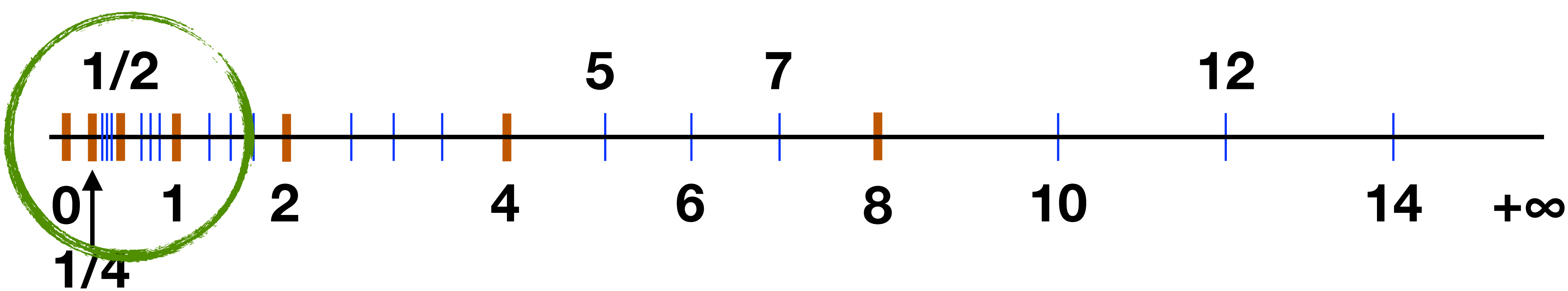
Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

0	001	11

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Uneven interval (c.f., fixed interval in fixed-point)
  - More dense toward 0, sparser toward infinite
  - Allow encoding small and large numbers at the same time

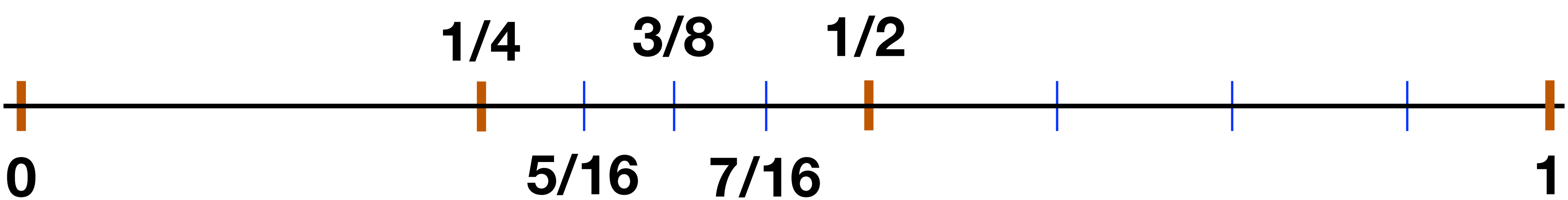


Representable Numbers (Positive Only)

$v = (-1)^s M\, 2^E$

s	exp	frac

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

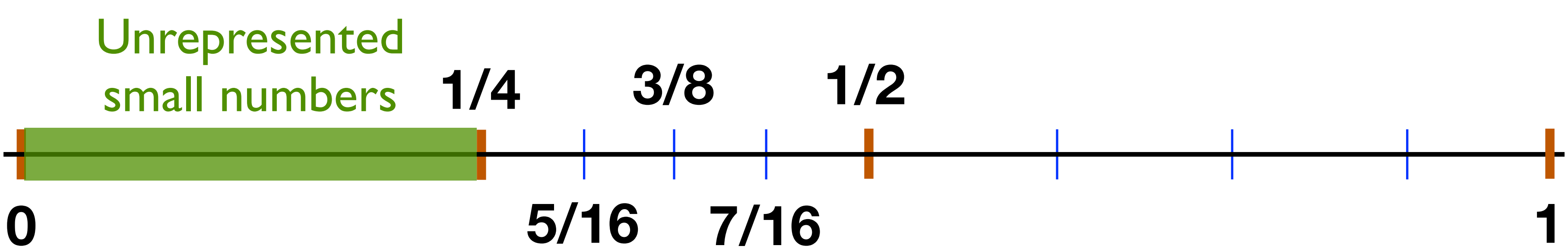


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

s	exp	frac

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

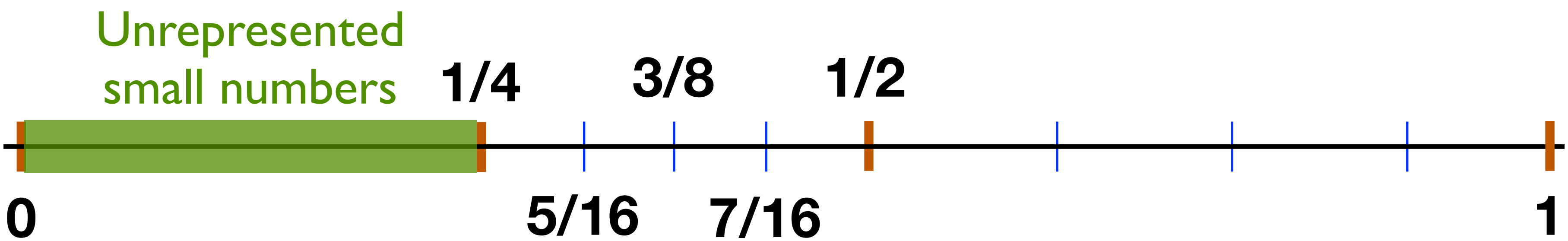


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Underflow: always round to 0 is inelegant



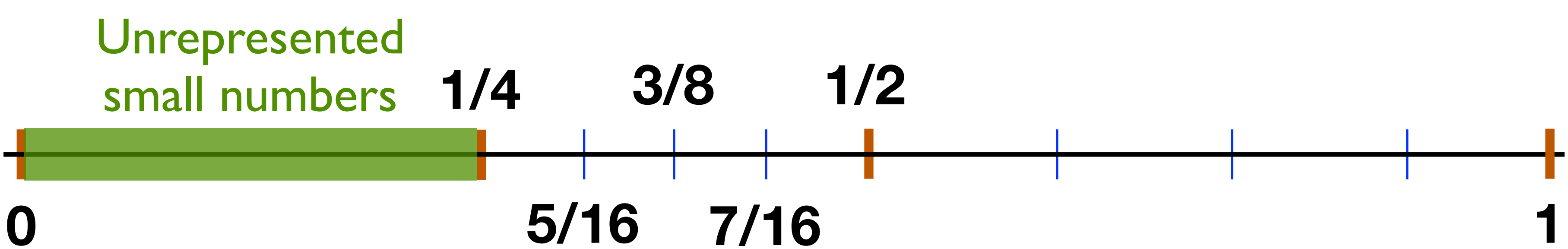
Representable Numbers (Positive Only)

$$v = (-1)^s \, M \, 2^E$$

s	exp	frac

E	exp	E	exp
-3	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Underflow: always round to 0 is inelegant

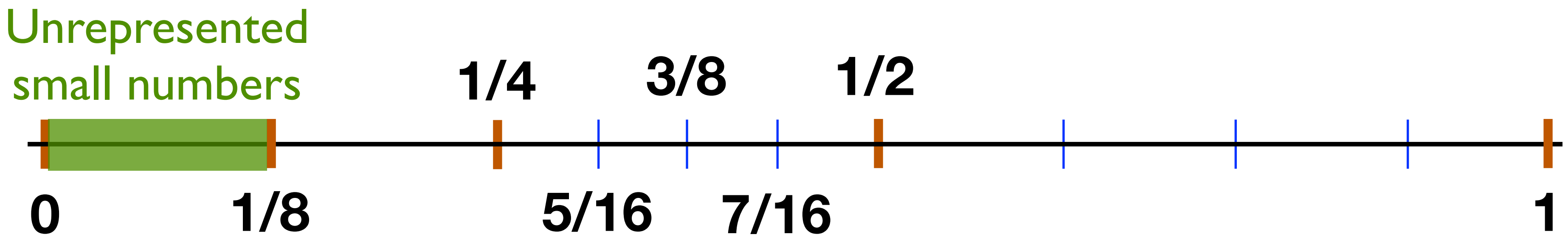


Representable Numbers (Positive Only)

$v = (-1)^s M 2^E$

E	exp	E	exp
-3	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Underflow: always round to 0 is inelegant



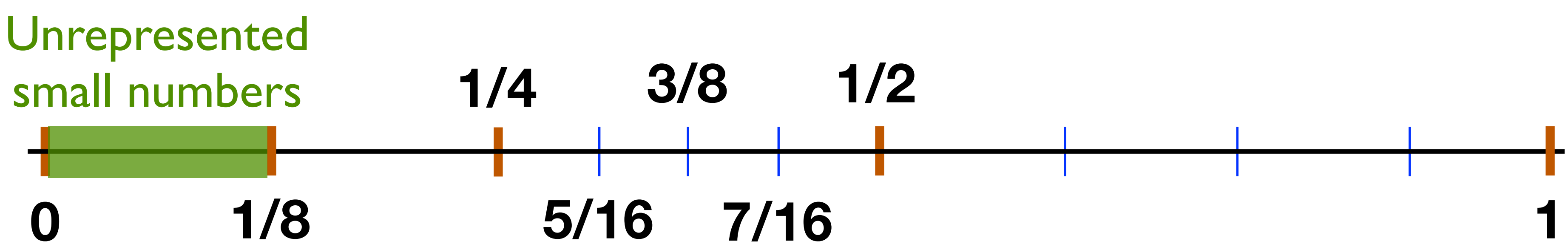
Representable Numbers (Positive Only)

$$v = (-1)^s M\, 2^E$$

s	exp	frac

E	exp	E	exp
-3	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Underflow: always round to 0 is inelegant
- Using 000 for *exp* would only postpone the problem rather than solving it

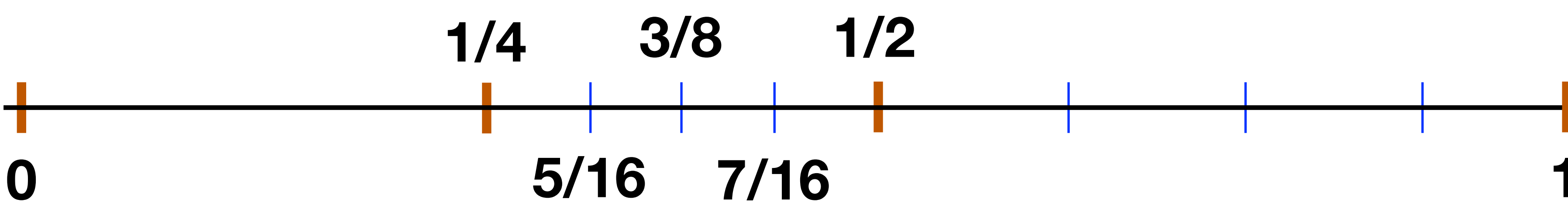


Subnormal (De-normalized) Numbers

$v = (-1)^s M 2^E$

E	exp	E	exp
-3	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Idea: Evenly divide between 0 and 1/4 rather than exponentially decreasing **when *exp* = 0** (subnormal numbers)



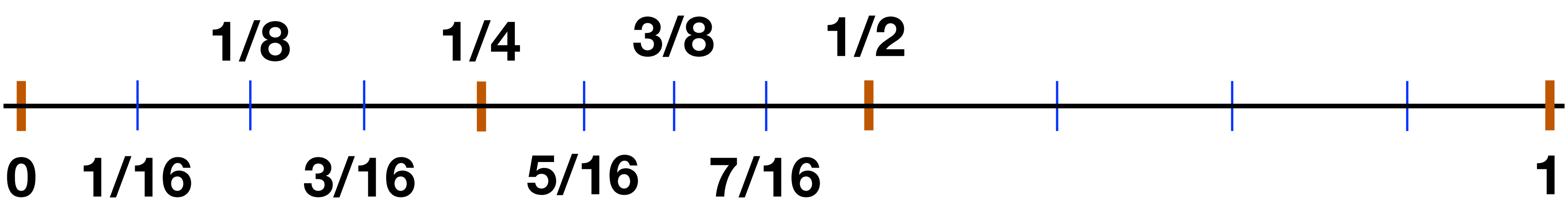
Subnormal (De-normalized) Numbers

$$v = (-1)^s \, M \, 2^E$$

s	exp	frac

E	exp	E	exp
-3	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Idea: Evenly divide between 0 and 1/4 rather than exponentially decreasing **when *exp* = 0** (subnormal numbers)



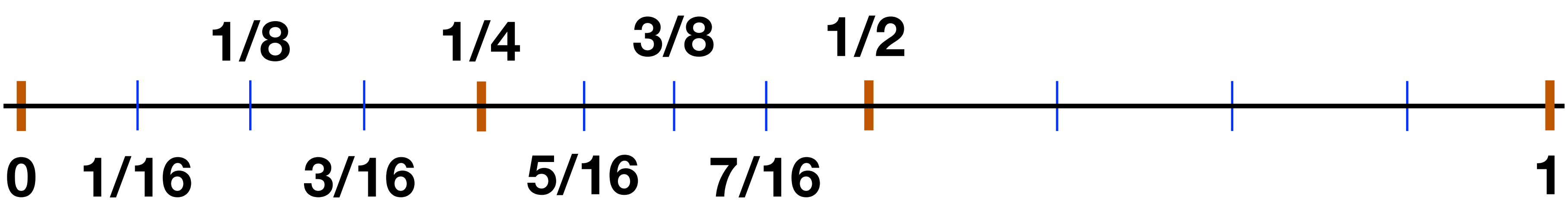
Subnormal (De-normalized) Numbers

$v = (-1)^s M \, 2^E$

s	exp	frac

E	exp	E	exp
-3	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Idea: Evenly divide between 0 and 1/4 rather than exponentially decreasing **when *exp* = 0** (subnormal numbers)
- $E = (exp + 1) - \text{bias}$ (instead of *exp* - *bias*)
- $M = 0.frac$ (instead of 1.frac)



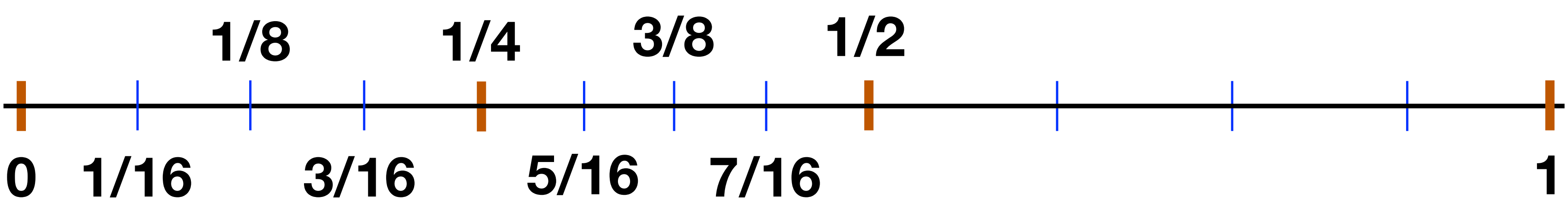
Subnormal (De-normalized) Numbers

$v = (-1)^s \, M \, 2^E$

s	exp	frac

E	exp	E	exp
-3	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Idea: Evenly divide between 0 and 1/4 rather than exponentially decreasing **when *exp* = 0** (subnormal numbers)
- $E = (exp + 1) - \text{bias}$ (instead of *exp* - *bias*)
- $M = 0.frac$ (instead of 1.frac)



0	000	01

$= (-1)^0 \, 0.01 \times 2^{(0+1-3)} = 1/16$

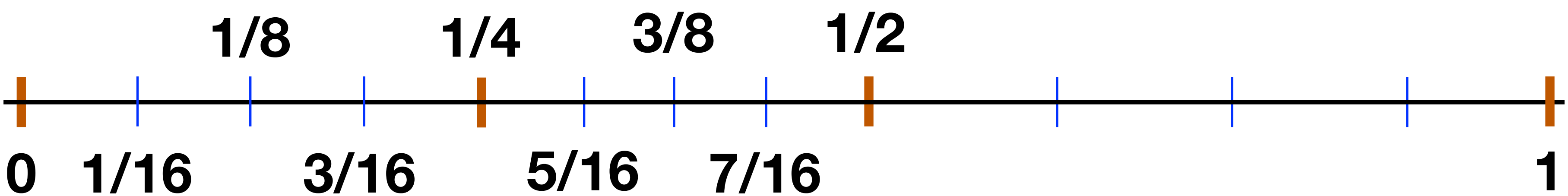
Subnormal (De-normalized) Numbers

$v = (-1)^s\, M\, 2^E$

s	exp	frac

E	exp	E	exp
-3	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- Idea: Evenly divide between 0 and 1/4 rather than exponentially decreasing **when *exp* = 0** (subnormal numbers)
- $E = (exp + 1) - \text{bias}$ (instead of *exp* - *bias*)
- $M = 0.frac$ (instead of 1.frac)
- Subnormal numbers allow graceful underflow



0	000	01

$= (-1)^0\, 0.01 \times 2^{(0+1-3)} = 1/16$

Special Values

$$v = (-1)^s M \, 2^E$$

E	exp	E	exp
-2	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

Special Values

$$v = (-1)^s \, M \, 2^E$$

s	exp	frac

E	exp	E	exp
-2	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- There are many special values in scientific computing
  - +/- ∞, NaNs (0 / 0, 0 / ∞, ∞ / ∞, …), etc.

Special Values

$v = (-1)^s \, M \, 2^E$

E	exp	E	exp
-2	000	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

- There are many special values in scientific computing
  - +/- ∞, NaNs (0 / 0, 0 / ∞, ∞ / ∞, …), etc.
- exp = 111 is reserved to represent these numbers

Special Values

$$v = (-1)^s M 2^E$$

E	exp	E	exp
-2	000	1	100
-2	001	2	101
-1	010	3	110
0	011		111

- There are many special values in scientific computing
  - +/- ∞, NaNs (0 / 0, 0 / ∞, ∞ / ∞, …), etc.
- exp = 111 is reserved to represent these numbers

Special Values

$v = (-1)^s M \, 2^E$

s	exp	frac

E	exp	E	exp
-2	000	1	100
-2	001	2	101
-1	010	3	110
0	011		111

- There are many special values in scientific computing
  - +/- ∞, NaNs (0 / 0, 0 / ∞, ∞ / ∞, …), etc.
- exp = 111 is reserved to represent these numbers
- exp = 111, frac = 000
  - +/- ∞ (depending on the *s* bit). Overflow results.
  - Arithmetic on ∞ is exact: 1.0/0.0 = –1.0/–0.0 = + ∞, 1.0/–0.0 = -∞

Special Values

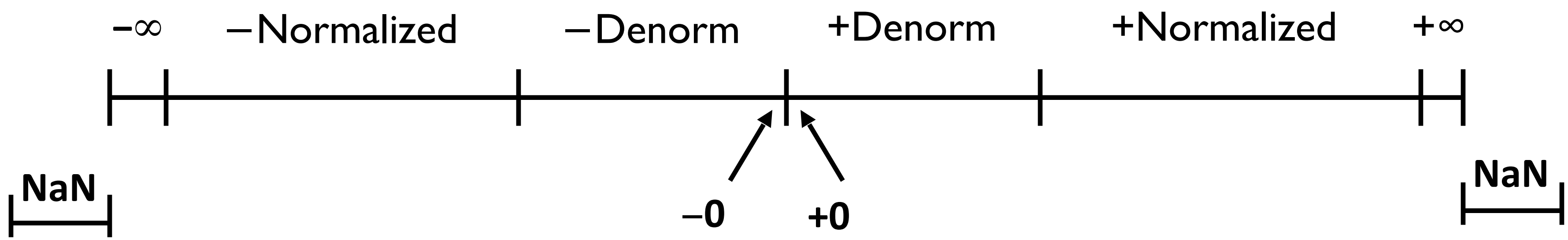
$$v = (-1)^s\, M\, 2^E$$

s	exp	frac

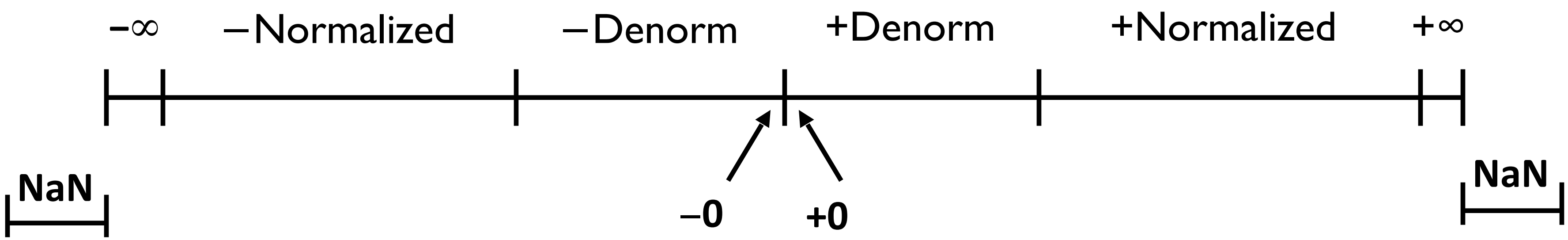
E	exp	E	exp
-2	000	1	100
-2	001	2	101
-1	010	3	110
0	011		111

- There are many special values in scientific computing
  - +/- ∞, NaNs (0 / 0, 0 / ∞, ∞ / ∞, …), etc.
- exp = 111 is reserved to represent these numbers
- exp = 111, frac = 000
  - +/- ∞ (depending on the *s* bit). Overflow results.
  - Arithmetic on ∞ is exact: 1.0/0.0 = –1.0/–0.0 = + ∞, 1.0/–0.0 = -∞
- exp = 111, frac != 000
  - Represent Not-a-Numbers (e.g., sqrt(–1), ∞ - ∞, ∞ x 0)

Visualization: Floating Point Encodings

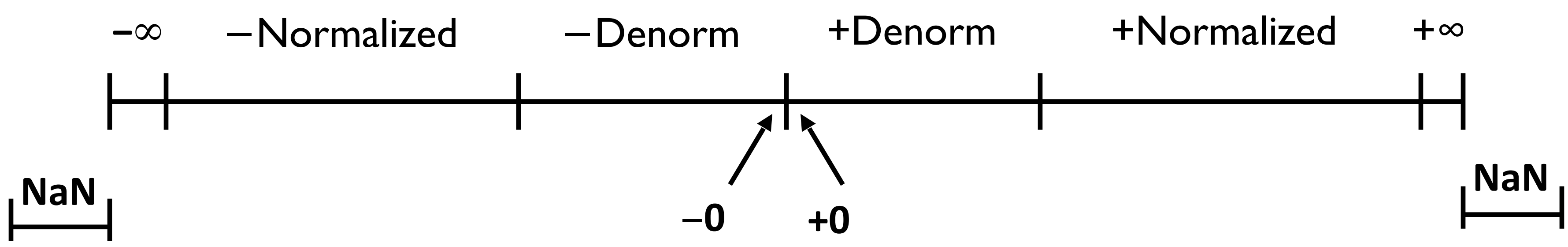


Visualization: Floating Point Encodings

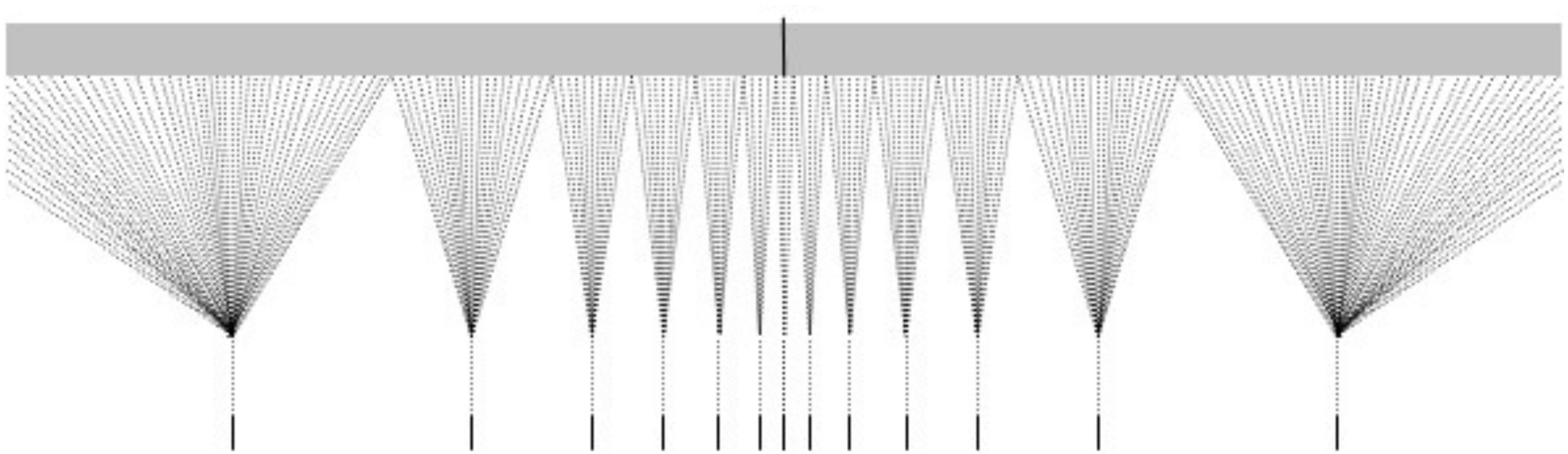


Infinite Amount of Real Numbers

Visualization: Floating Point Encodings

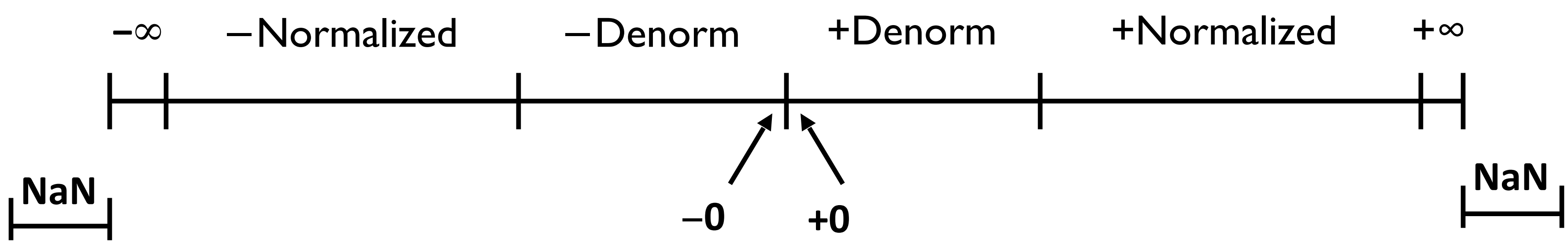


Infinite Amount of Real Numbers

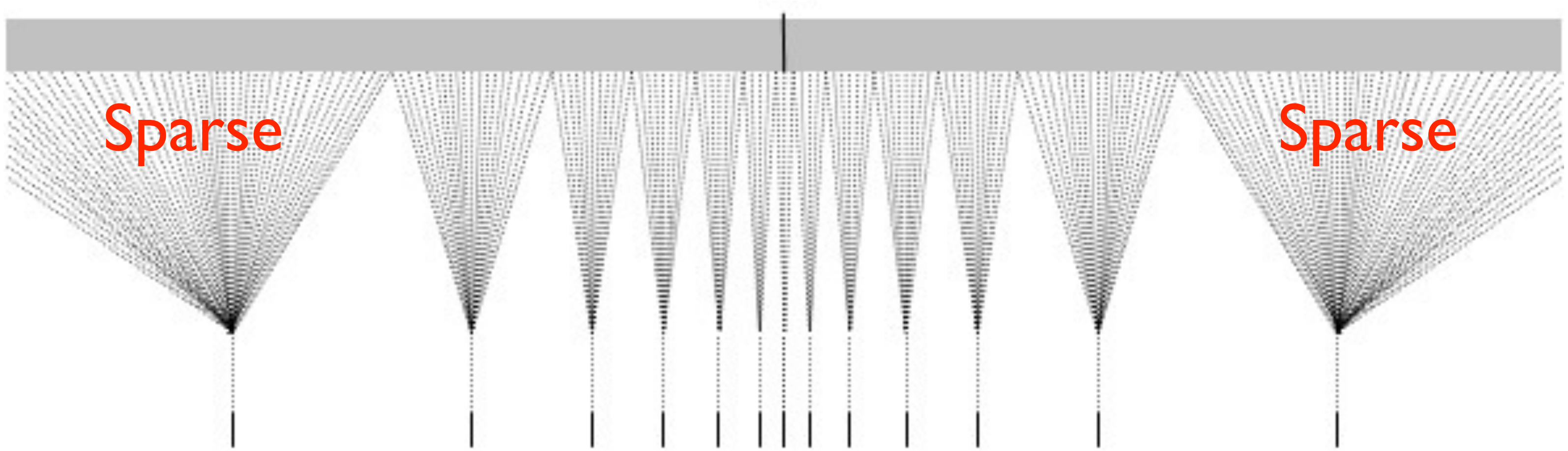


Finite Amount of Floating Point Numbers

Visualization: Floating Point Encodings

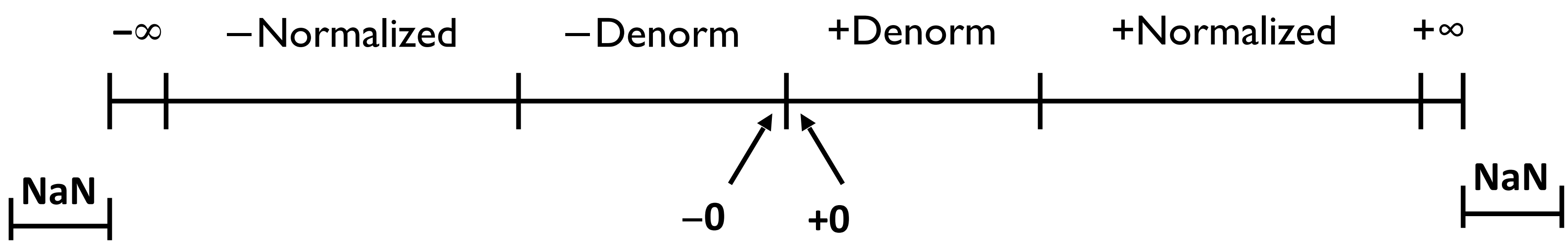


Infinite Amount of Real Numbers

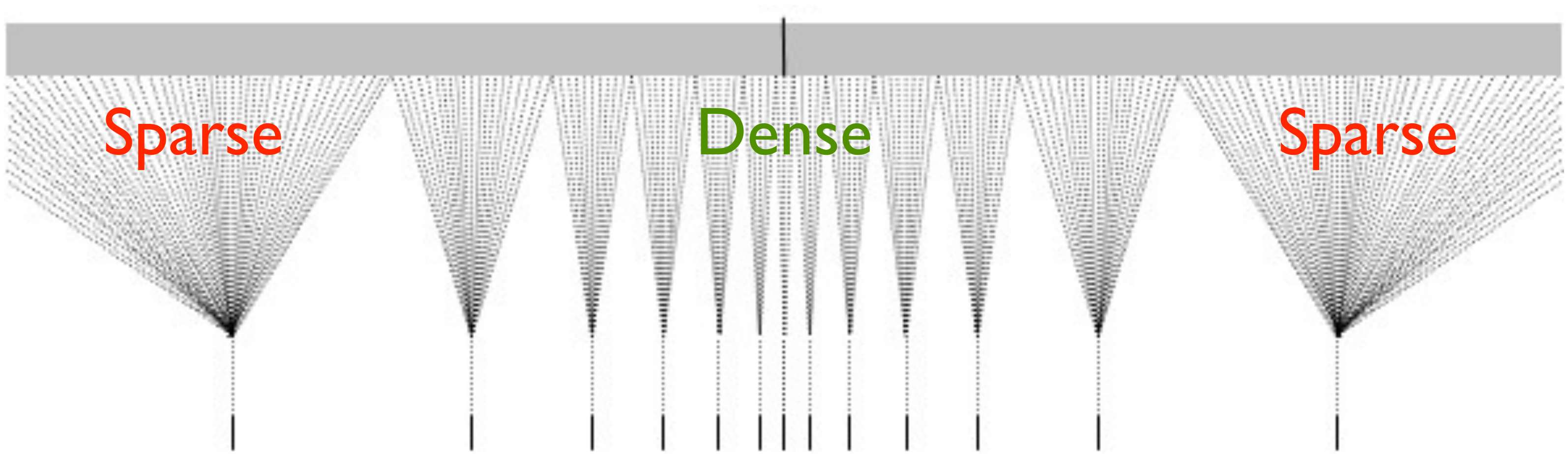


Finite Amount of Floating Point Numbers

Visualization: Floating Point Encodings

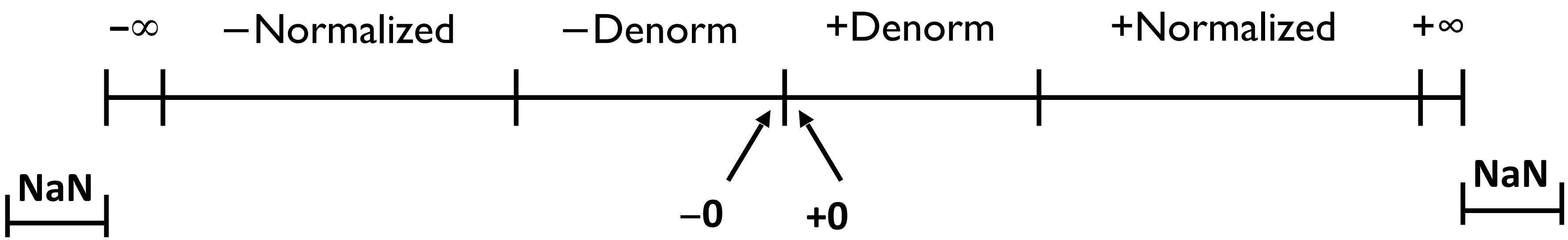


Infinite Amount of Real Numbers

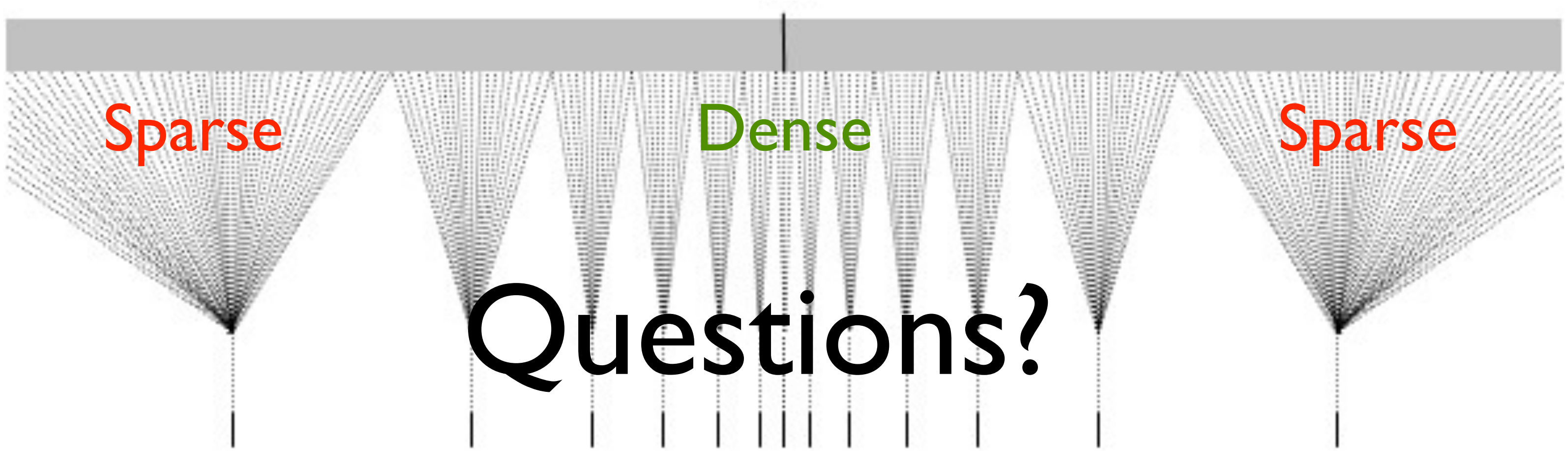


Finite Amount of Floating Point Numbers

Visualization: Floating Point Encodings



Infinite Amount of Real Numbers



Questions?

Finite Amount of Floating Point Numbers

Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

IEEE Floating Point

- IEEE Standard 754
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs (and even GPUs and other processors)

- Driven by numerical concerns
  - Nice standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard

IEEE 754 Standard Precision Options

- Single precision: 32 bits

- Double precision: 64 bits

Single Precision (32-bit) Example

$v = (-1)^s \, M \, 2^E$

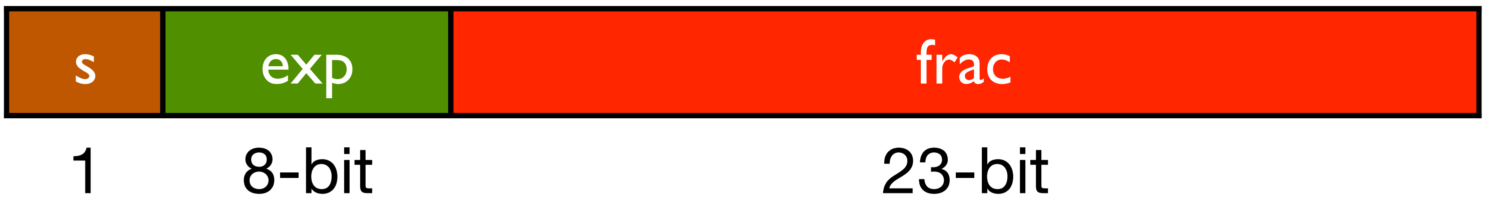
bias = $2^{(8-1)}-1 = 127$

s	exp	frac
1	8-bit	23-bit

Single Precision (32-bit) Example

$v = (-1)^s \, M \, 2^E$

bias = $2^{(8-1)}-1 = 127$



$$15213_{10} = 11101101101101_2$$
$$= (-1)^0 \, 1.1101101101101_2 \times 2^{13}$$

Single Precision (32-bit) Example

$v = (-1)^s M\, 2^E$

bias $= 2^{(8-1)}-1 = 127$

0	exp	frac
1	8-bit	23-bit

$$15213_{10} = 11101101101101_2$$
$$= (-1)^0\, 1.1101101101101_2 \times 2^{13}$$

Single Precision (32-bit) Example

$v = (-1)^s M 2^E$

$\text{bias} = 2^{(8-1)}-1 = 127$

0	exp	frac
1	8-bit	23-bit

$$15213_{10} = 11101101101101_2$$
$$= (-1)^0 \, 1.1101101101101_2 \times 2^{\boxed{13}}$$

Single Precision (32-bit) Example

$v = (-1)^s \, M \, 2^E$

$\text{bias} = 2^{(8-1)}-1 = 127$

0	exp	frac
1	8-bit	23-bit

$$15213_{10} = 11101101101101_2$$

$$= (-1)^0 \, 1.1101101101101_2 \times 2^{\boxed{13}}$$

$\text{exp} = E + \text{bias} = 140_{10}$

Single Precision (32-bit) Example

$v = (-1)^s M\, 2^E$

bias $= 2^{(8-1)}-1 = 127$

0	10001100	frac
1	8-bit	23-bit

$$15213_{10} = 11101101101101_2$$
$$= (-1)^0\, 1.1101101101101_2 \times 2^{\boxed{13}}$$

exp = E + bias = 140_{10}

Single Precision (32-bit) Example

$v = (-1)^s \, M \, 2^E$

bias = $2^{(8-1)}-1 = 127$

0	10001100	frac
1	8-bit	23-bit

$$15213_{10} = 11101101101101_2$$

$$= (-1)^0 \, 1.\boxed{1101101101101}_2 \times 2^{\boxed{13}}$$

exp = E + bias = 140_{10}

Single Precision (32-bit) Example

$v = (-1)^s \, M \, 2^E$

bias $= 2^{(8-1)}-1 = 127$

0	10001100	11011011011010000000000
1	8-bit	23-bit

$$15213_{10} = 11101101101101_2$$

$$= (-1)^0 \, 1.\boxed{1101101101101}_2 \times 2^{\boxed{13}}$$

exp = E + bias = 140_{10}

Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

Floating Point Operations: Basic Idea

- Basic idea
  - We perform the operation & produce the infinitely precise result
  - Make it fit into desired precision
    - Possibly overflow if exponent too large
    - Possibly round to fit into frac

- $x +_f y = \mathrm{Round}(x + y)$

- $x \times_f y = \mathrm{Round}(x \times y)$

Rounding Modes

Rounding Modes

- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)

Rounding Modes

- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)
- Directed rounding:
  - Towards zero (chop)
  - Round down (-∞)
  - Round up (+∞)

Rounding Modes

- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)
- Directed rounding:
  - Towards zero (chop)
  - Round down (-∞)
  - Round up (+∞)

Rounding Mode	**1.40**	**1.60**	**1.50**	**2.50**	**-1.50**
Towards zero	1	1	1	2	-1
Round down (-∞)	1	1	1	2	-2
Round up (+∞)	2	2	2	3	-1
Nearest even (default)	1	2	2	2	-2

Rounding Modes (Binary Example)

- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*

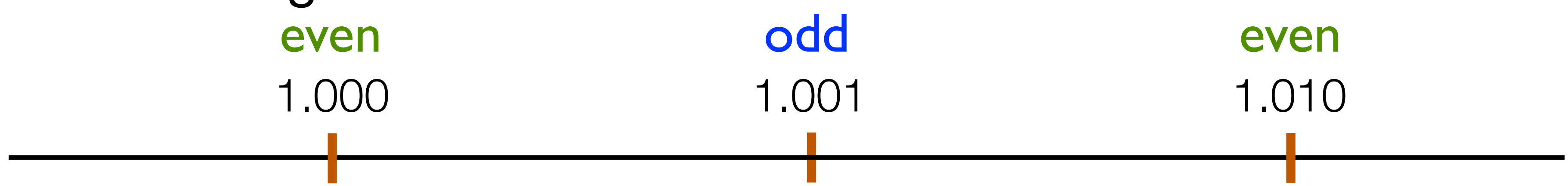
Rounding Modes (Binary Example)

- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*

Precise Value	Rounded Value	Notes
1.000011	1.000	1.000 is the nearest (down)
1.000110	1.001	1.001 is the nearest (up)
1.000100	1.000	1.000 is the nearest even (down)
1.001100	1.010	1.010 is the nearest even (up)

Rounding Modes (Binary Example)

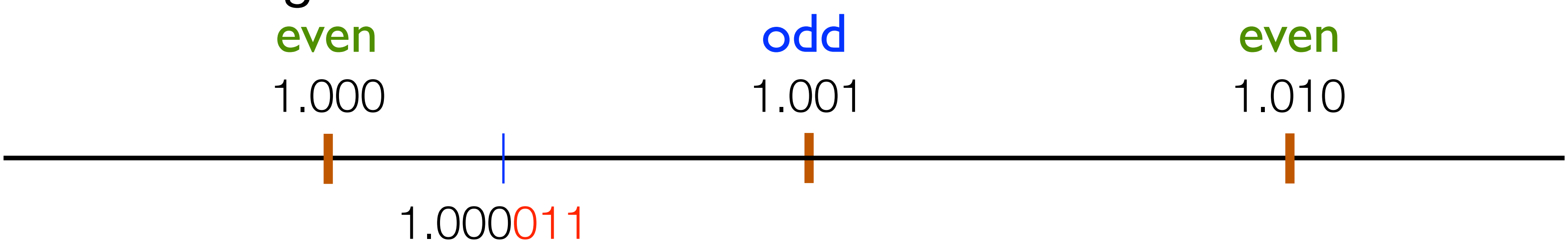
- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



Precise Value	Rounded Value	Notes
1.000011	1.000	1.000 is the nearest (down)
1.000110	1.001	1.001 is the nearest (up)
1.000100	1.000	1.000 is the nearest even (down)
1.001100	1.010	1.010 is the nearest even (up)

Rounding Modes (Binary Example)

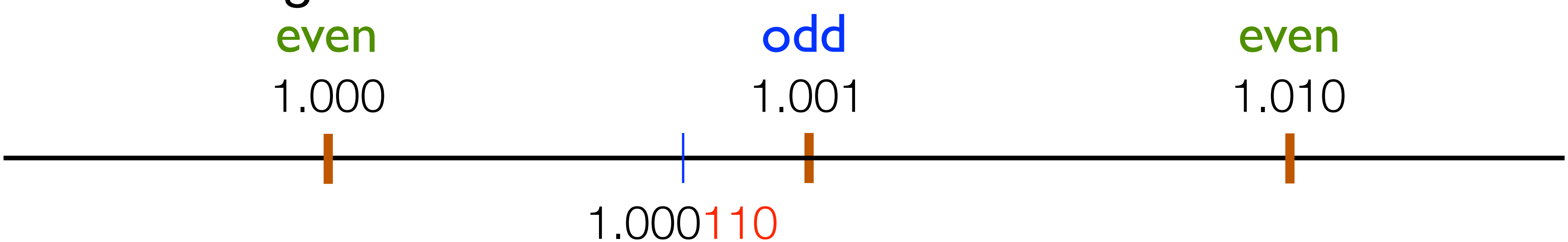
- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



Precise Value	**Rounded Value**	**Notes**
1.000011	1.000	1.000 is the nearest (down)
1.000110	1.001	1.001 is the nearest (up)
1.000100	1.000	1.000 is the nearest even (down)
1.001100	1.010	1.010 is the nearest even (up)

Rounding Modes (Binary Example)

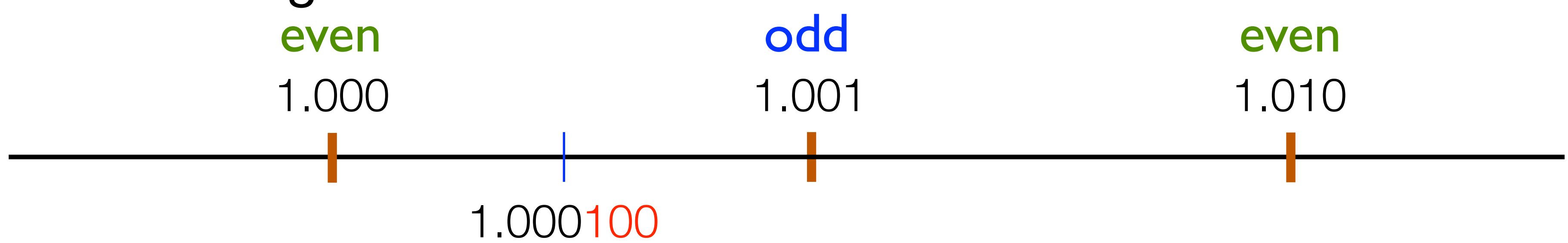
- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



Precise Value	**Rounded Value**	**Notes**
1.000011	1.000	1.000 is the nearest (down)
1.000110	1.001	1.001 is the nearest (up)
1.000100	1.000	1.000 is the nearest even (down)
1.001100	1.010	1.010 is the nearest even (up)

Rounding Modes (Binary Example)

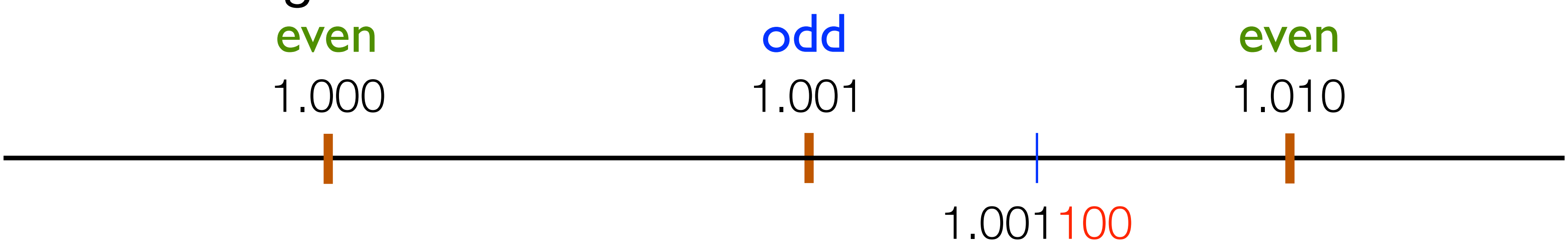
- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



Precise Value	Rounded Value	Notes
1.000011	1.000	1.000 is the nearest (down)
1.000110	1.001	1.001 is the nearest (up)
1.000100	1.000	1.000 is the nearest even (down)
1.001100	1.010	1.010 is the nearest even (up)

Rounding Modes (Binary Example)

- Default: To nearest; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



Precise Value	Rounded Value	Notes
1.000011	1.000	1.000 is the nearest (down)
1.000110	1.001	1.001 is the nearest (up)
1.000100	1.000	1.000 is the nearest even (down)
1.001100	1.010	1.010 is the nearest even (up)

Floating Point Addition

Floating Point Addition

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$

$1.000 \times 2^{-1} + 11.10 \times 2^{-3}$

Floating Point Addition

- $(-1)^{s1}\ M1\ \ 2^{E1} \quad + \quad (-1)^{s2}\ M2\ \ 2^{E2}$

$$1.000 \times 2^{-1} + 11.10 \times 2^{-3}$$

$$\Downarrow$$

align $\quad 1.000 \times 2^{-1} + 0.111 \times 2^{-1}$

Floating Point Addition

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$

$$1.000 \times 2^{-1} + 11.10 \times 2^{-3}$$

$$\downarrow$$

$$1.000 \times 2^{-1} + 0.111 \times 2^{-1}$$

$$\downarrow$$

add $\quad 1.111 \times 2^{-1}$

Floating Point Addition

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign *s*, significand *M*:
    - Result of signed align & add
  - Exponent E: E1
    - Assume E1 > E2

$1.000 \times 2^{-1} + 11.10 \times 2^{-3}$

$1.000 \times 2^{-1} + 0.111 \times 2^{-1}$

1.111×2^{-1}

Floating Point Addition

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s, significand M:
    - Result of signed align & add
  - Exponent E: E1
    - Assume E1 > E2
- Fixing
  - If $M \geq 2$, shift M right, increment E
  - If $M < 1$, shift M left k positions, decrement E by k
  - Overflow if E out of range
  - Round M to fit *frac* precision

$$1.000 \times 2^{-1} + 11.10 \times 2^{-3}$$

$$1.000 \times 2^{-1} + 0.111 \times 2^{-1}$$

$$1.111 \times 2^{-1}$$

Mathematical Properties of FP Add

Mathematical Properties of FP Add

- Commutative?

Mathematical Properties of FP Add

- Commutative? ***Yes***

Mathematical Properties of FP Add

- Commutative? **Yes**
- Associative?

Mathematical Properties of FP Add

- Commutative? ***Yes***
- Associative? ***No***
  - Overflow and inexactness of rounding
  - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`

Mathematical Properties of FP Add

- Commutative? **Yes**
- Associative? **No**
  - Overflow and inexactness of rounding
  - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
- 0 is additive identity?

Mathematical Properties of FP Add

- Commutative? **Yes**
- Associative? **No**
  - Overflow and inexactness of rounding
  - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
- 0 is additive identity? **Yes**

Mathematical Properties of FP Add

- Commutative? **Yes**
- Associative? **No**
  - Overflow and inexactness of rounding
  - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
- 0 is additive identity? **Yes**
- Every element has additive inverse (negation)?

Mathematical Properties of FP Add

- Commutative? ***Yes***
- Associative? ***No***
  - Overflow and inexactness of rounding
  - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
- 0 is additive identity? ***Yes***
- Every element has additive inverse (negation)? ***Almost***
  - Except for infinities & NaNs

Mathematical Properties of FP Add

- Commutative? ***Yes***
- Associative? ***No***
  - Overflow and inexactness of rounding
  - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
- 0 is additive identity? ***Yes***
- Every element has additive inverse (negation)? ***Almost***
  - Except for infinities & NaNs
- Monotonicity: $a \geq b \Rightarrow a+c \geq b+c$?

Mathematical Properties of FP Add

- Commutative? ***Yes***
- Associative? ***No***
  - Overflow and inexactness of rounding
  - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
- 0 is additive identity? ***Yes***
- Every element has additive inverse (negation)? ***Almost***
  - Except for infinities & NaNs
- Monotonicity: $a \geq b \Rightarrow a+c \geq b+c$? ***Almost***
  - Except for infinities & NaNs

Floating Point Multiplication

Floating Point Multiplication

- $(-1)^{s1}\ M1\ \ 2^{E1}$ x $(-1)^{s2}\ M2\ \ 2^{E2}$

Floating Point Multiplication

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \ \text{x}\ \ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign *s*: $s1 \wedge s2$
  - Significand *M*: $M1 \text{ x } M2$
  - Exponent *E*: $E1 + E2$

Floating Point Multiplication

- $(-1)^{s1}\ M1\ \ 2^{E1}\quad \times \quad (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s: $s1 \wedge s2$
  - Significand M: $M1 \times M2$
  - Exponent E: $E1 + E2$
- Fixing
  - If M ≥ 2, shift M right, increment E
  - If E out of range, overflow
  - Round M to fit frac precision

Floating Point Multiplication

- $(-1)^{s1}\ M1\ \ 2^{E1}\quad \text{x}\quad (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s: $s1$ ^ $s2$
  - Significand M: $M1$ x $M2$
  - Exponent E: $E1 + E2$
- Fixing
  - If M ≥ 2, shift M right, increment E
  - If E out of range, overflow
  - Round M to fit frac precision
- Implementation
  - Biggest chore is multiplying significands

Mathematical Properties of FP Mult

Mathematical Properties of FP Mult

- Multiplication Commutative?

Mathematical Properties of FP Mult

- Multiplication Commutative? **Yes**

Mathematical Properties of FP Mult

- Multiplication Commutative? **Yes**
- Multiplication is Associative?

Mathematical Properties of FP Mult

- Multiplication Commutative? **Yes**
- Multiplication is Associative? **No**
  - Possibility of overflow, inexactness of rounding
  - Ex: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`

Mathematical Properties of FP Mult

- Multiplication Commutative? **Yes**
- Multiplication is Associative? **No**
  - Possibility of overflow, inexactness of rounding
  - Ex: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
- 1 is multiplicative identity?

Mathematical Properties of FP Mult

- Multiplication Commutative? **Yes**
- Multiplication is Associative? **No**
  - Possibility of overflow, inexactness of rounding
  - Ex: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
- 1 is multiplicative identity? **Yes**

Mathematical Properties of FP Mult

- Multiplication Commutative? **Yes**
- Multiplication is Associative? **No**
  - Possibility of overflow, inexactness of rounding
  - Ex: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
- 1 is multiplicative identity? **Yes**
- Multiplication distributes over addition?

Mathematical Properties of FP Mult

- Multiplication Commutative? **Yes**
- Multiplication is Associative? **No**
  - Possibility of overflow, inexactness of rounding
  - Ex: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
- 1 is multiplicative identity? **Yes**
- Multiplication distributes over addition? **No**
  - Possibility of overflow, inexactness of rounding
  - `1e20*(1e20-1e20)= 0.0,  1e20*1e20 – 1e20*1e20 = NaN`

Mathematical Properties of FP Mult

- Multiplication Commutative? **Yes**
- Multiplication is Associative? **No**
  - Possibility of overflow, inexactness of rounding
  - Ex: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
- 1 is multiplicative identity? **Yes**
- Multiplication distributes over addition? **No**
  - Possibility of overflow, inexactness of rounding
  - `1e20*(1e20-1e20)= 0.0,  1e20*1e20 - 1e20*1e20 = NaN`
- Monotonicity: $a \geq b$ & $c \geq 0 \Rightarrow a * c \geq b * c$?

Mathematical Properties of FP Mult

- Multiplication Commutative? **Yes**
- Multiplication is Associative? **No**
  - Possibility of overflow, inexactness of rounding
  - Ex: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
- 1 is multiplicative identity? **Yes**
- Multiplication distributes over addition? **No**
  - Possibility of overflow, inexactness of rounding
  - `1e20*(1e20-1e20)= 0.0,  1e20*1e20 – 1e20*1e20 = NaN`
- Monotonicity: $a \geq b$ & $c \geq 0 \Rightarrow a * c \geq b * c$? ***Almost***
  - Except for infinities & NaNs

Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

Floating Point in C

64-bit Machine

	C Data Type	Bits	Max Value	Max Value (Decimal)
Fixed point (implicit binary point)	`char`	8	$2^7 - 1$	127
	`short`	16	$2^{15} - 1$	32767
	`int`	32	$2^{31} - 1$	2147483647
	`long`	64	$2^{31} - 1$	$\sim 9.2 \times 10^{18}$
SP floating point	`float`	32	$(2 - 2^{-23}) \times 2^{127}$	$\sim 3.4 \times 10^{38}$
DP floating point	`double`	64	$(2 - 2^{-52}) \times 2^{1023}$	$\sim 1.8 \times 10^{308}$

Floating Point in C

- C Guarantees Two Levels
  - `float` single precision
  - `double` double precision
- Conversions/Casting
  - Casting between **`int`**, **`float`**, and **`double`** changes bit representation
  - **`double`**/**`float`** → **`int`**
    - Truncates fractional part
    - Like rounding toward zero
    - Not defined when out of range or NaN: Generally sets to TMin
  - **`int`** → **`double`**
    - Exact conversion, as long as int has ≤ 53 bit word size
  - **`int`** → **`float`**
    - Will round according to rounding mode