CSC 252/452: Computer Organization
Fall 2024: Lecture 4

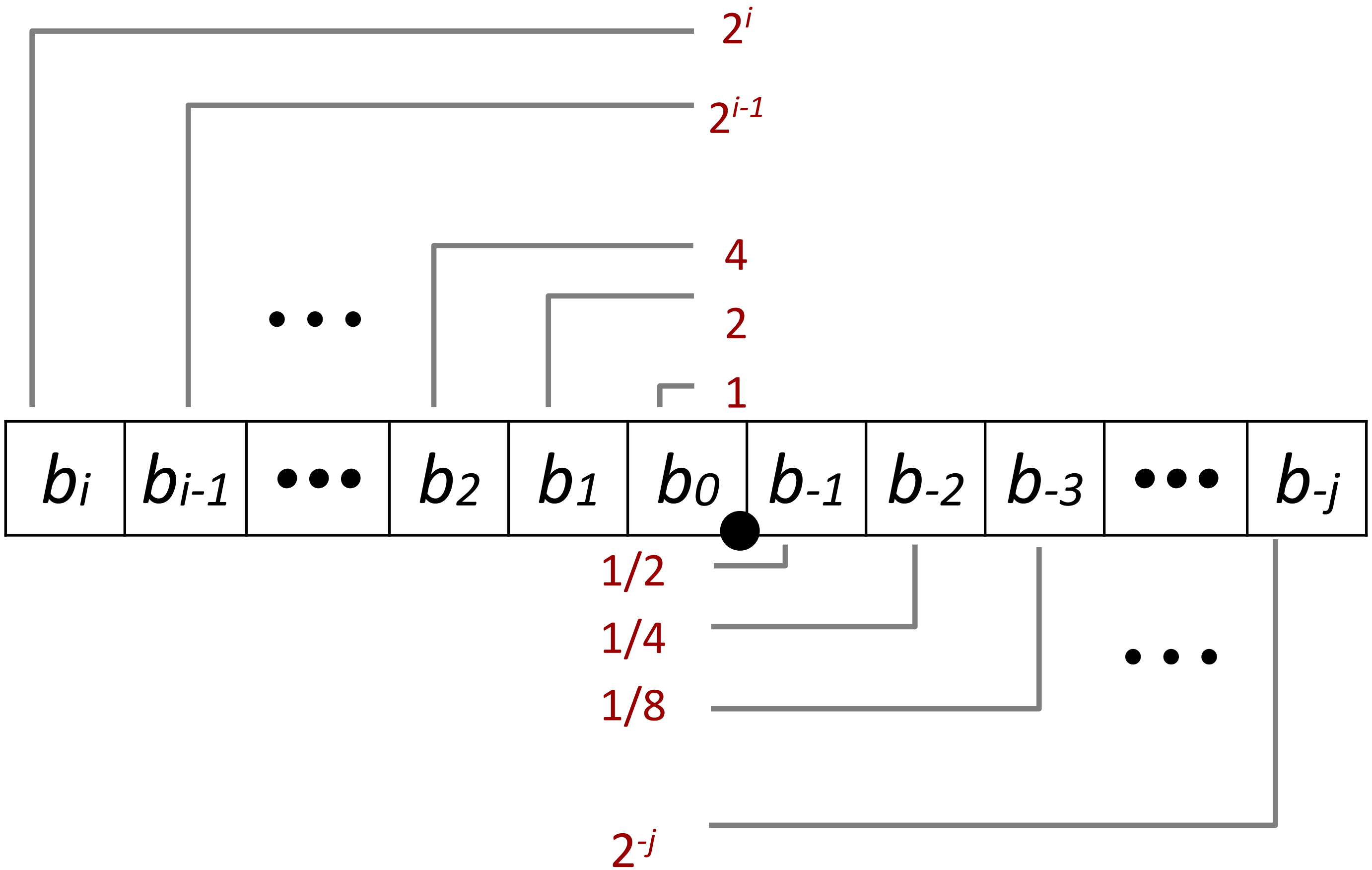
Instructor: Yanan Guo

Department of Computer Science
University of Rochester

Announcement

- Programming Assignment 1 is out
  - Details: https://www.cs.rochester.edu/courses/252/fall2024/labs/assignment1.html
  - Due on Sep. 16th, 11:59 PM
  - You have 3 slip days
- Office Hour Location Has Changed
  - For office hours on Thursday and Friday
  - Check course website

Fractional Binary Numbers



Fixed-Point Representation

- Binary point stays fixed
- Fixed interval between representable numbers
    - The interval in this example is 0.25_{10}

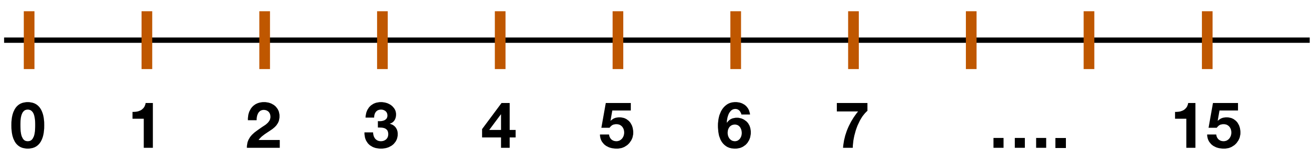
Fixed-Point Representation

- Binary point stays fixed
- Fixed interval between representable numbers
  - The interval in this example is 0.25_{10}

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

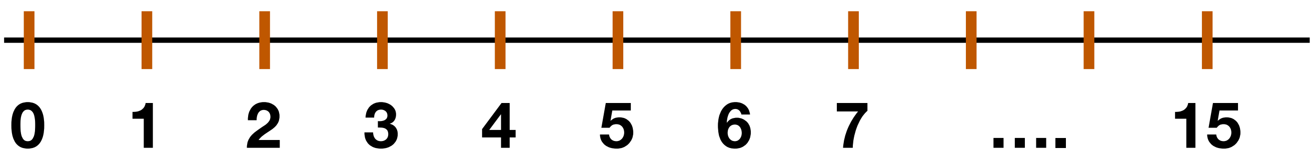
- Binary point stays fixed
- Fixed interval between representable numbers
  - The interval in this example is 0.25_{10}



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

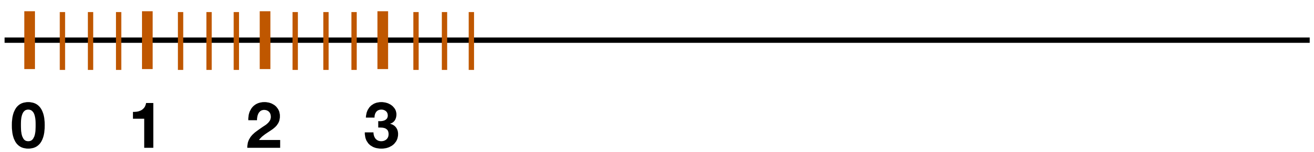
- Binary point stays fixed
- Fixed interval between representable numbers
  - The interval in this example is 0.25_{10}



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
  - The interval in this example is 0.25_{10}

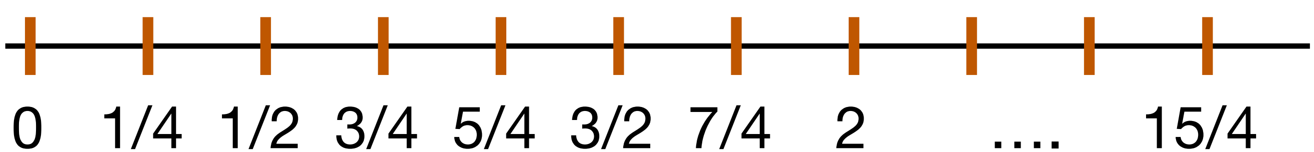


Decimal	Binary
0	00.00
0.25	00.01
0.5	00.10
0.75	00.11
1	01.00
1.25	01.01
1.5	01.10
1.75	01.11
2	10.00
2.25	10.01
2.5	10.10
2.75	10.11
3	11.00
3.25	11.01
3.5	11.10
3.75	11.11

Limitations of Fixed-Point (#*1*)

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

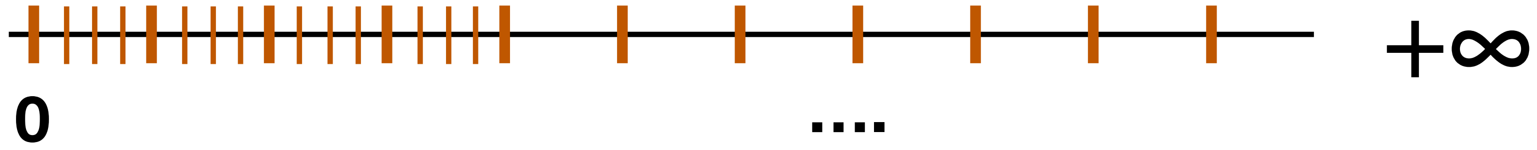
Decimal Value	**Binary Representation**
1/3	0.0101010101[01]…
1/5	0.001100110011[0011]…
1/10	0.0001100110011[0011]…



$b_3b_2.b_1b_0$

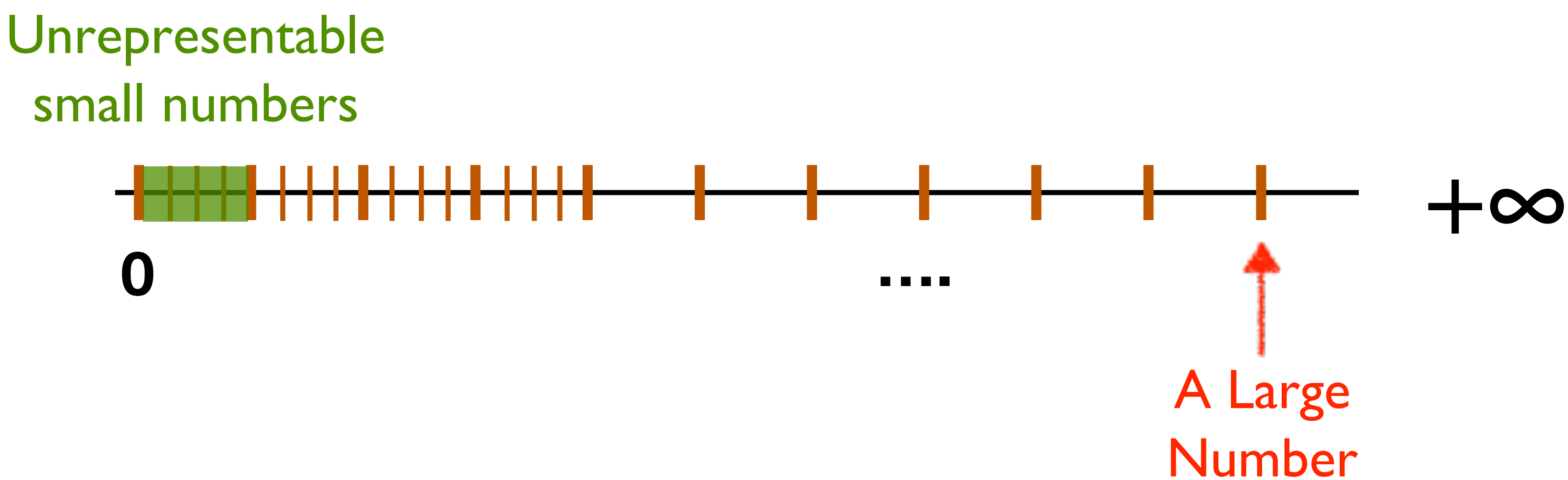
Limitations of Fixed-Point (#2)

- 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



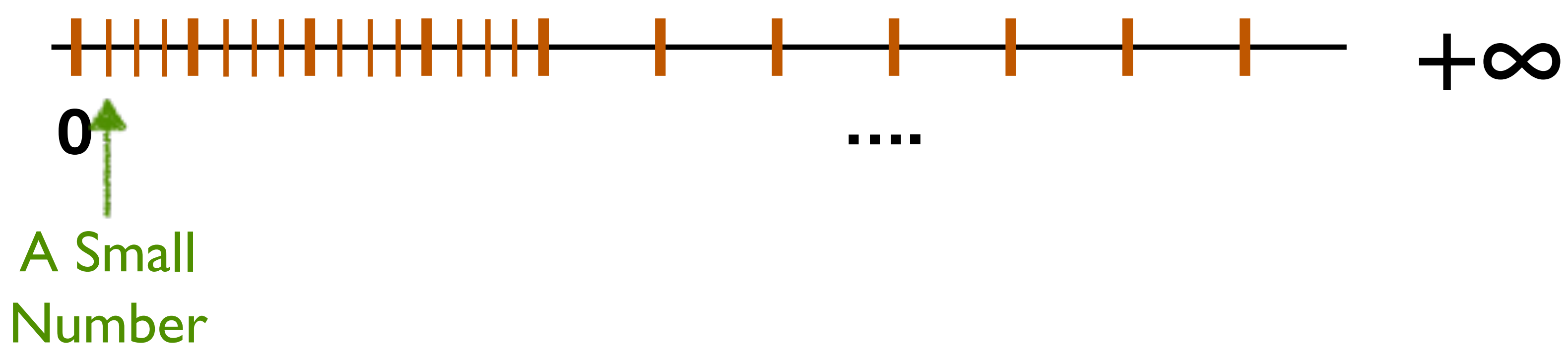
Limitations of Fixed-Point (#2)

- 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



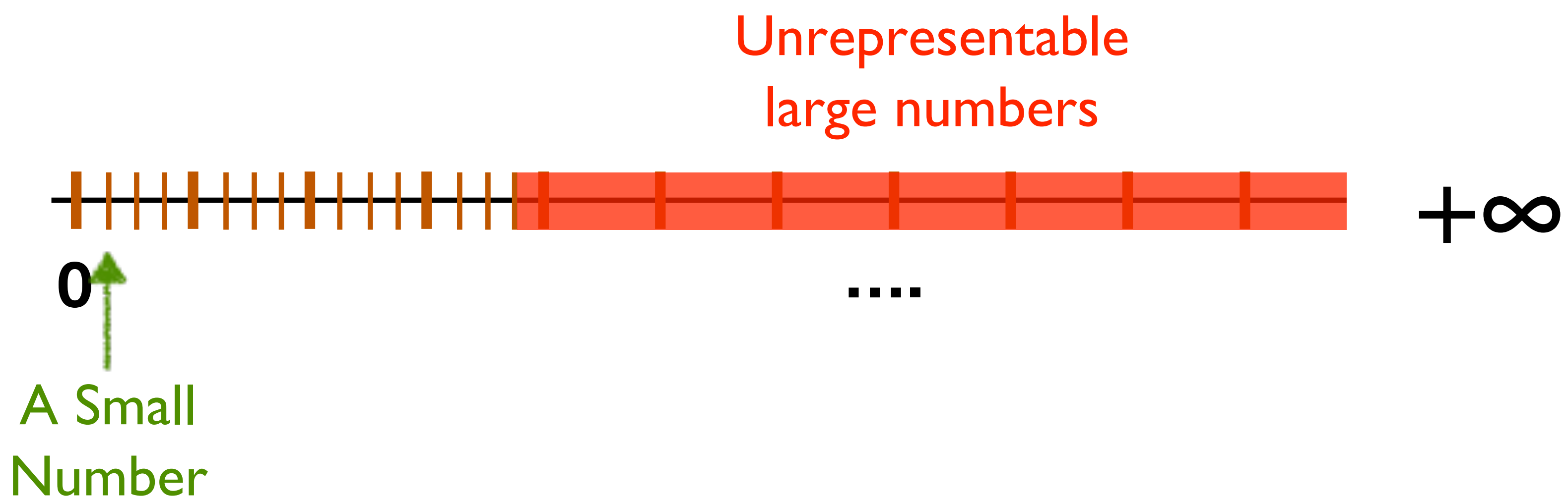
Limitations of Fixed-Point (#2)

- 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



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



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
  - MSB *s* is sign bit s
  - *exp* field encodes Exponent (but not exactly the same, more later)
  - *frac* field encodes Fraction (but not exactly the same, more later)

Sign Exponent

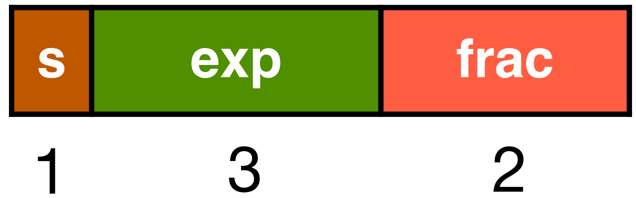
$$(-1)^{s}\, M \times 2^{E}$$

Significand Base

s	exp	frac

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?
  - Subtract 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?
  - Subtract 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 we use 3 bits for exp (i.e., 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

- *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

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

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:

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

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	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	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:

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

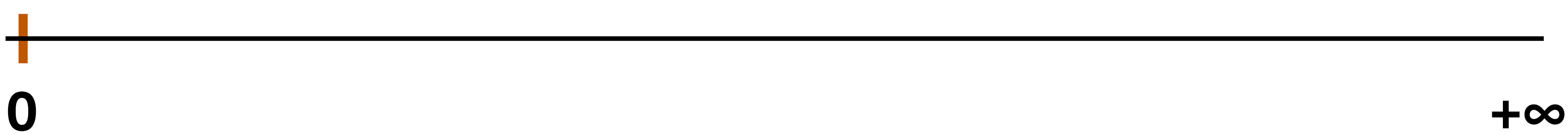
E	exp
~~-3~~	~~000~~
-2	001
-1	010
0	011
1	100
2	101
3	110
~~4~~	~~111~~

Representable Numbers (Positive Only)

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

0	001	00
1	3	2

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
1	3	2

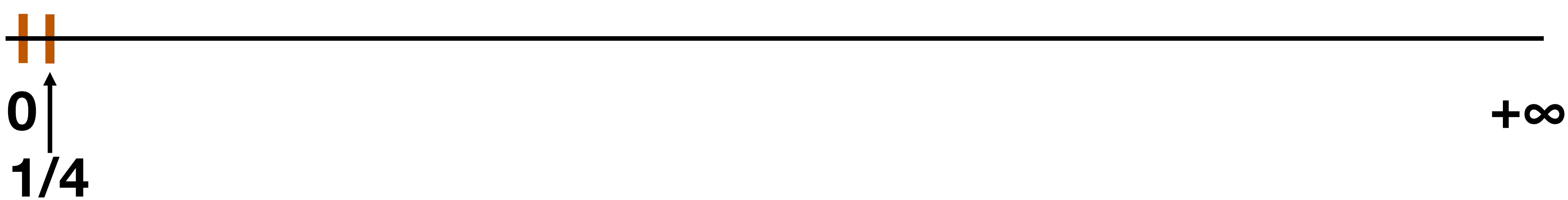
E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~

Representable Numbers (Positive Only)

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

0	001	00
1	3	2

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

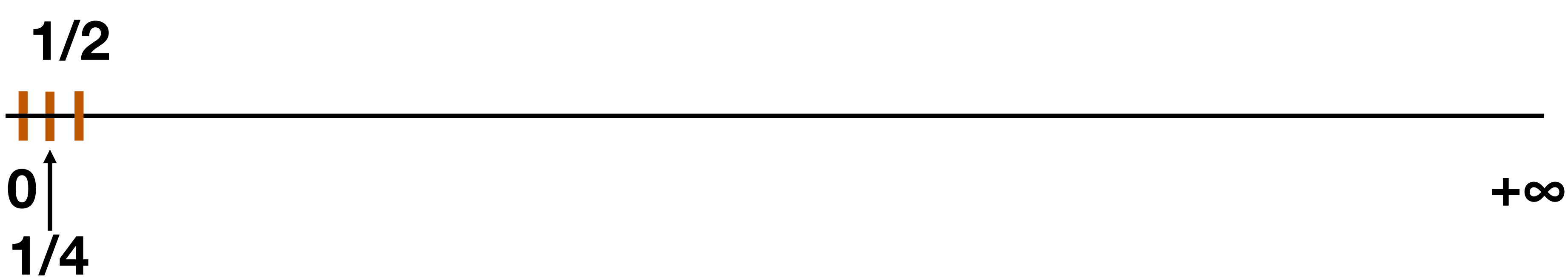


Representable Numbers (Positive Only)

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

0	010	00
1	3	2

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

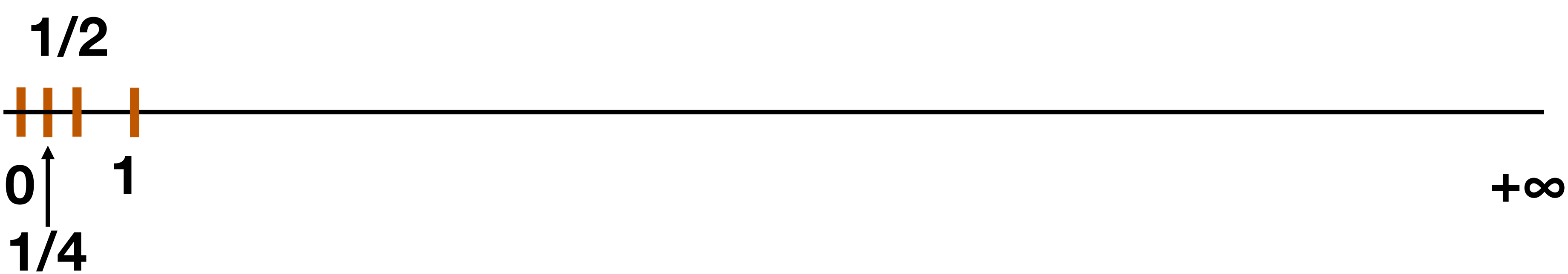


Representable Numbers (Positive Only)

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

0	011	00
1	3	2

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

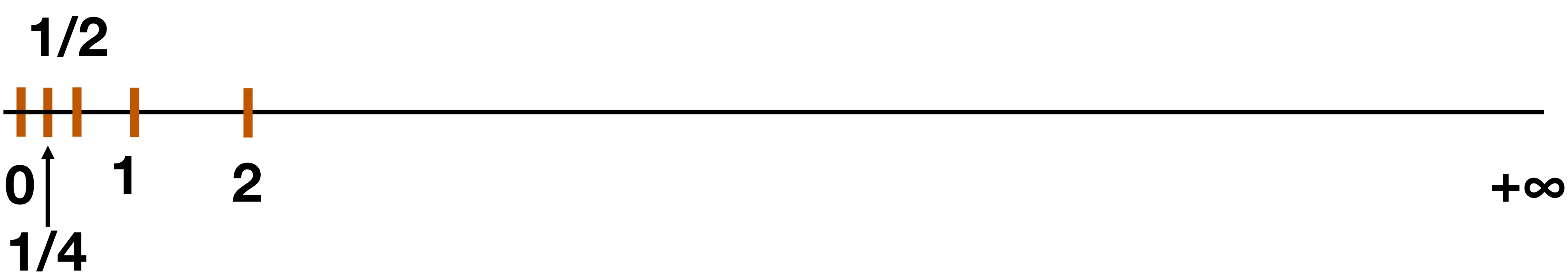


Representable Numbers (Positive Only)

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

0	100	00
1	3	2

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

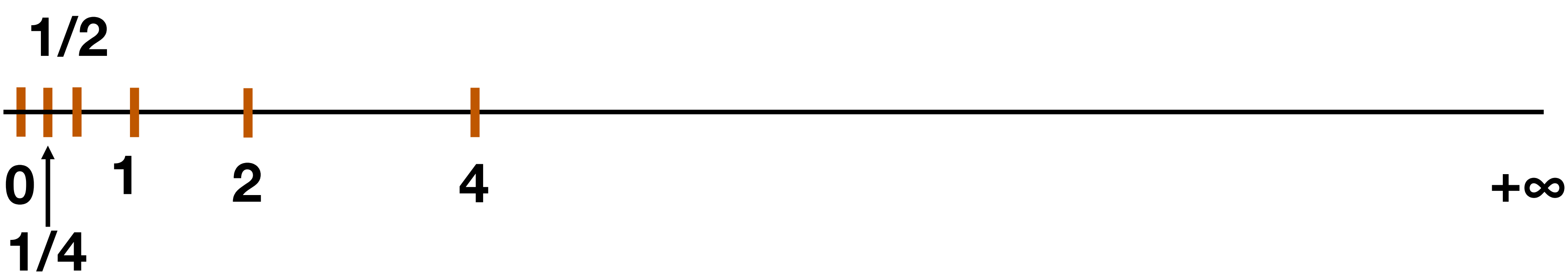


Representable Numbers (Positive Only)

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

0	101	00
1	3	2

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

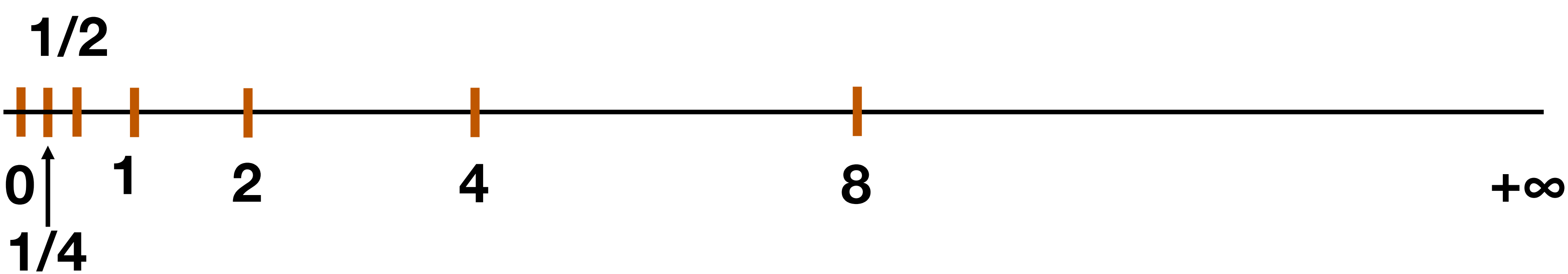


Representable Numbers (Positive Only)

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

0	110	00
1	3	2

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

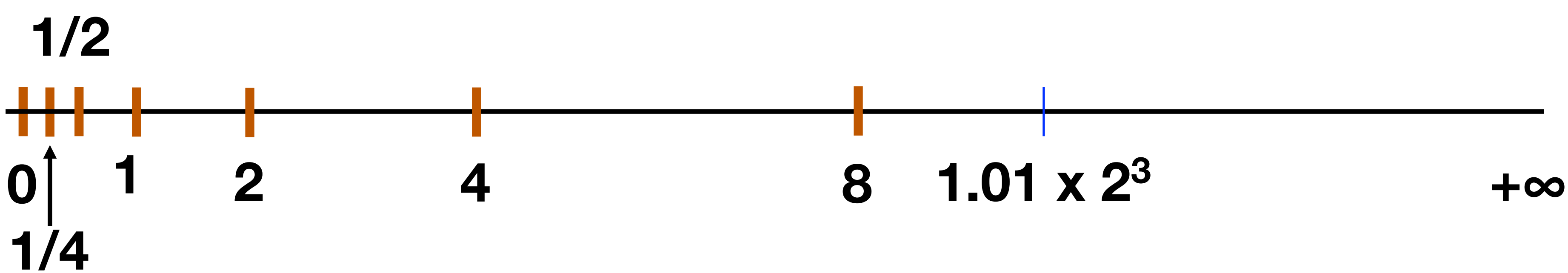


Representable Numbers (Positive Only)

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

0	110	01
1	3	2

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

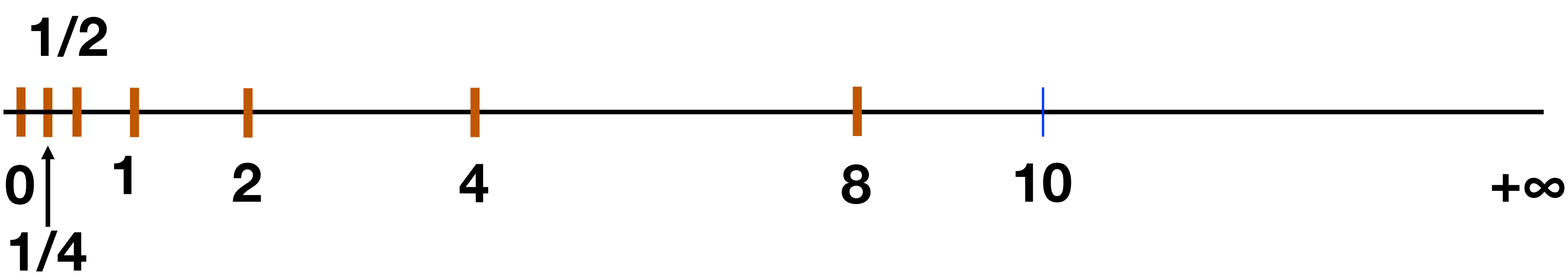


Representable Numbers (Positive Only)

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

s	exp	frac
0	110	01
1	3	2

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

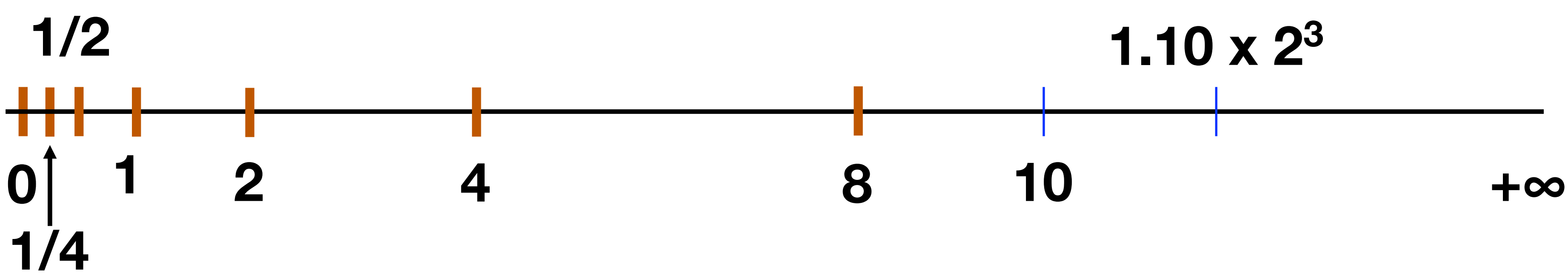


Representable Numbers (Positive Only)

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

0	110	10
1	3	2

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

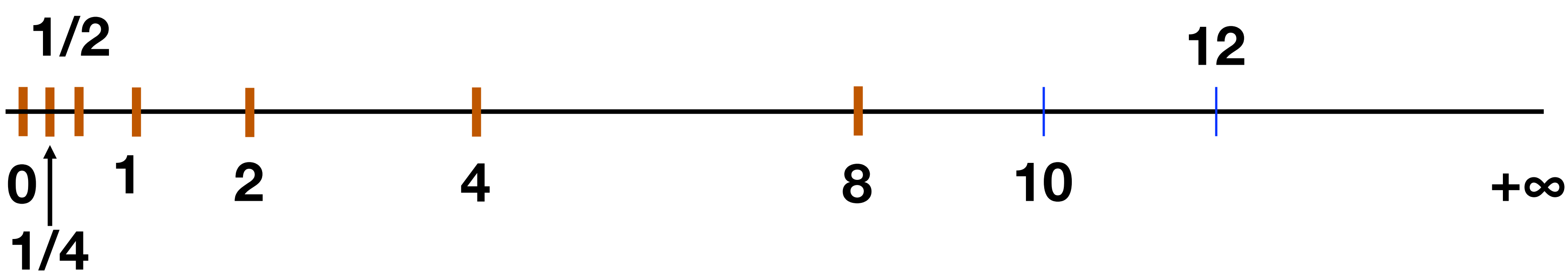


Representable Numbers (Positive Only)

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

0	110	10
1	3	2

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

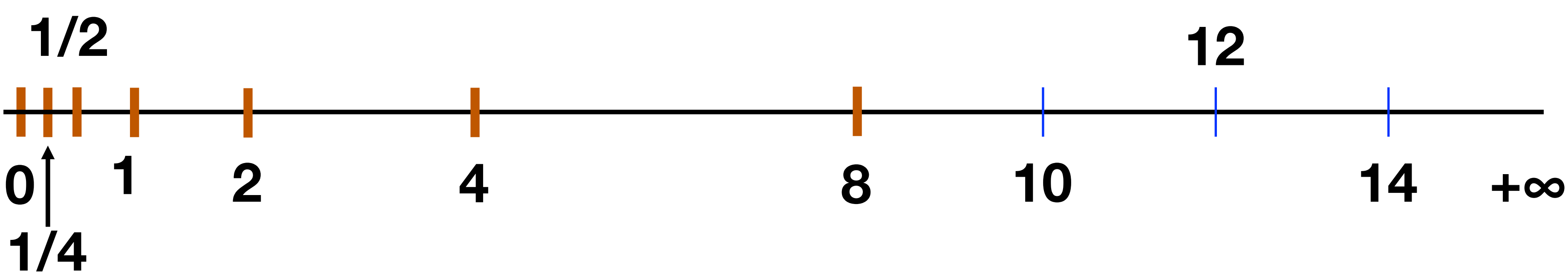


Representable Numbers (Positive Only)

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

0	110	11
1	3	2

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

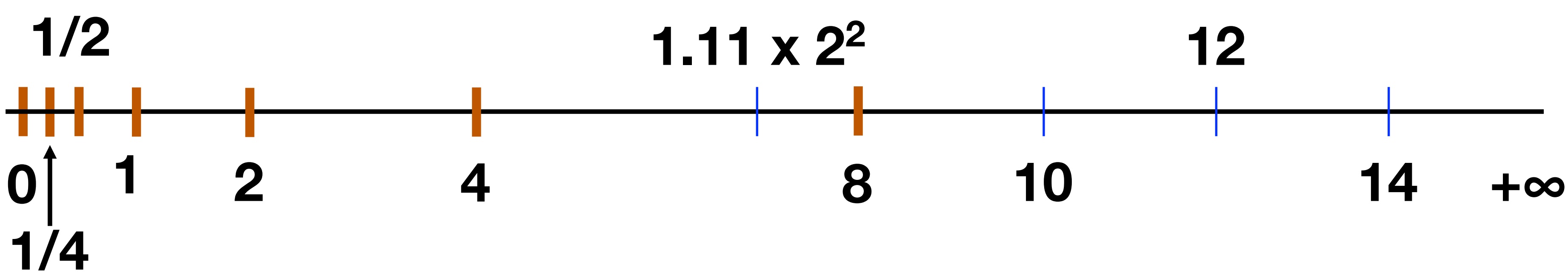


Representable Numbers (Positive Only)

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

0	101	11
1	3	2

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

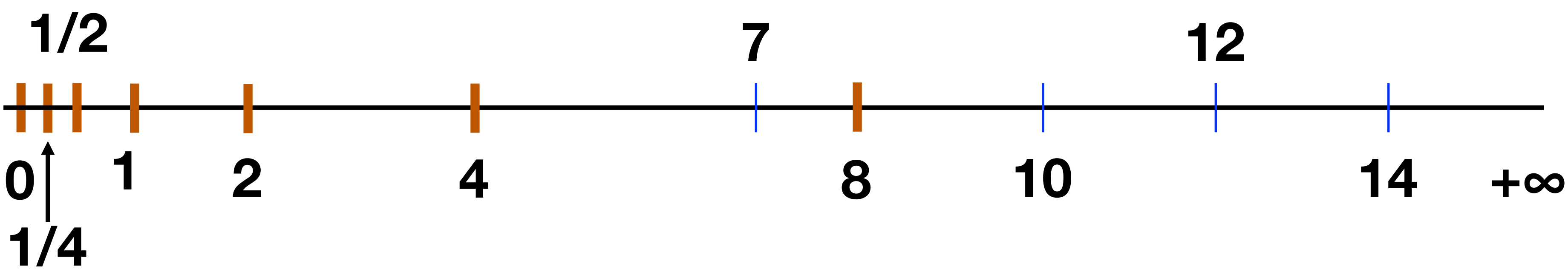


Representable Numbers (Positive Only)

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

0	101	11
1	3	2

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

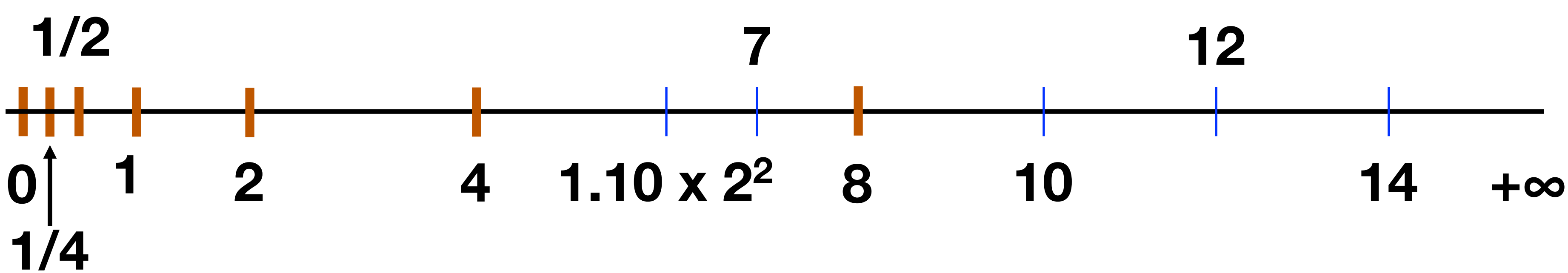


Representable Numbers (Positive Only)

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

0	101	10
1	3	2

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

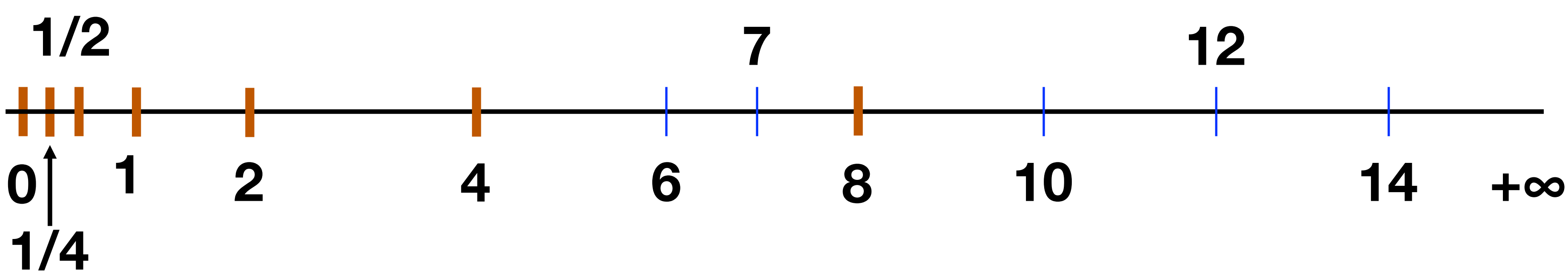


Representable Numbers (Positive Only)

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

0	101	10
1	3	2

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

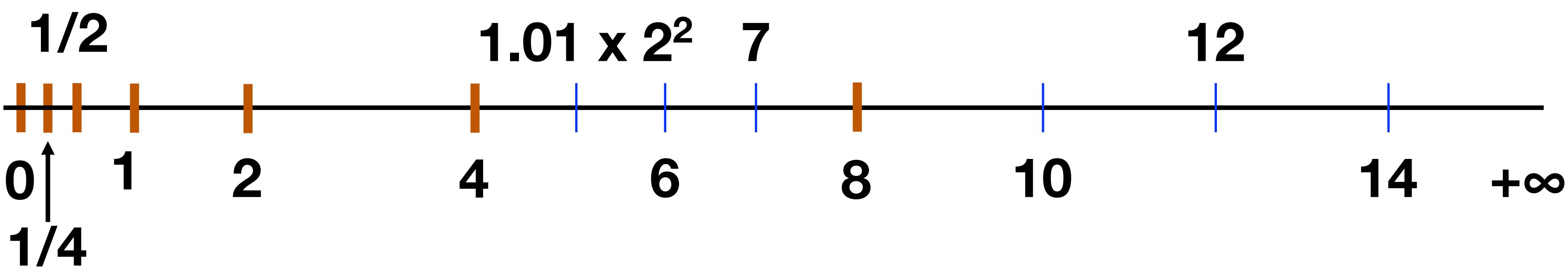


Representable Numbers (Positive Only)

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

0	101	01
1	3	2

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

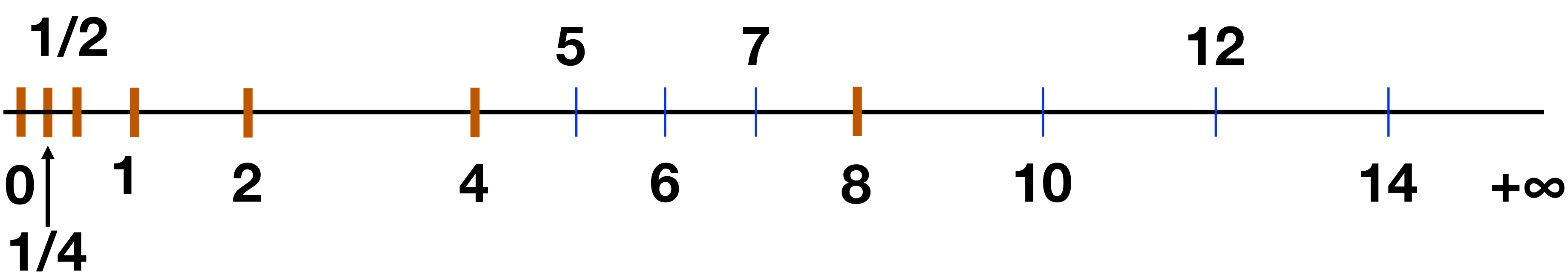


Representable Numbers (Positive Only)

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

0	101	01
1	3	2

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

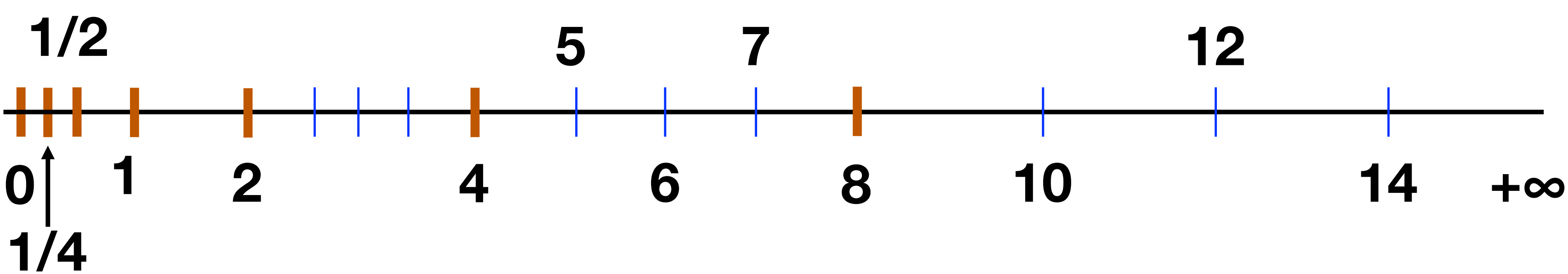


Representable Numbers (Positive Only)

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

0	100	frac
1	3	2

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

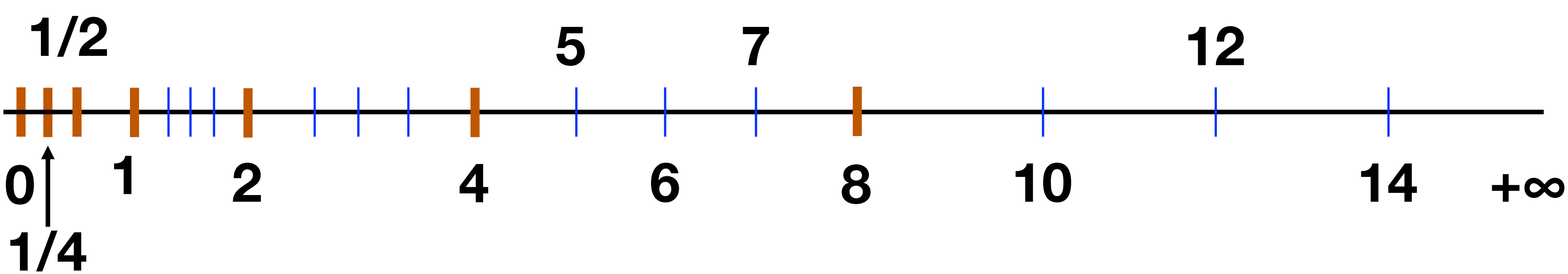


Representable Numbers (Positive Only)

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

0	011	frac
1	3	2

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

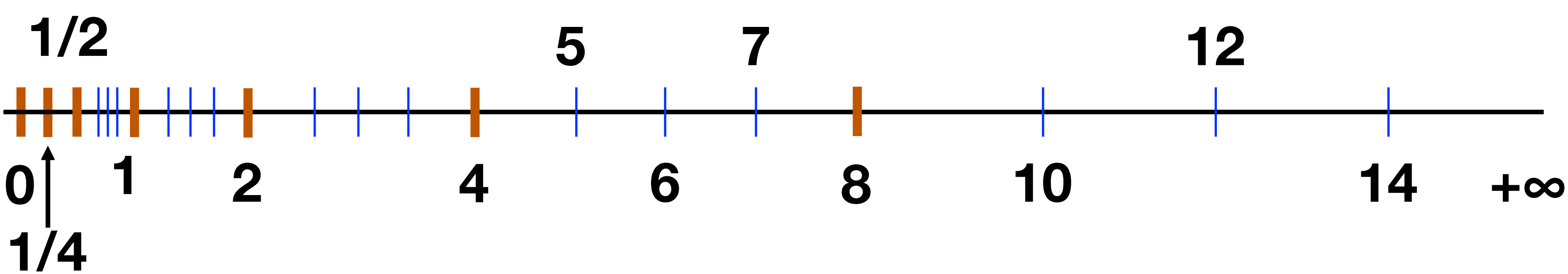


Representable Numbers (Positive Only)

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

0	010	frac
1	3	2

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

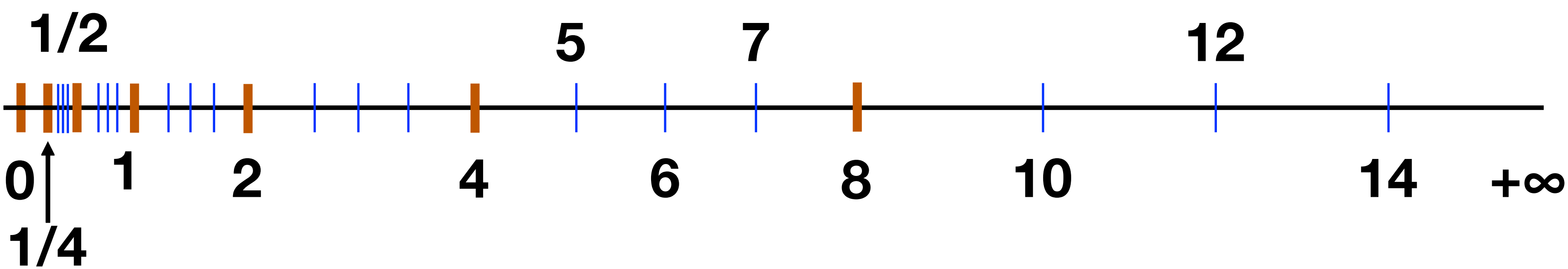


Representable Numbers (Positive Only)

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

0	001	frac
1	3	2

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

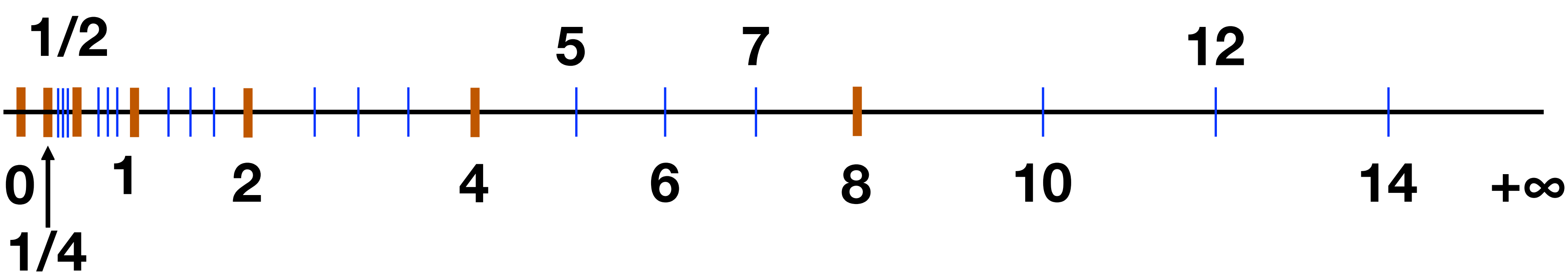


Representable Numbers (Positive Only)

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

s	exp	frac
1	3	2

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

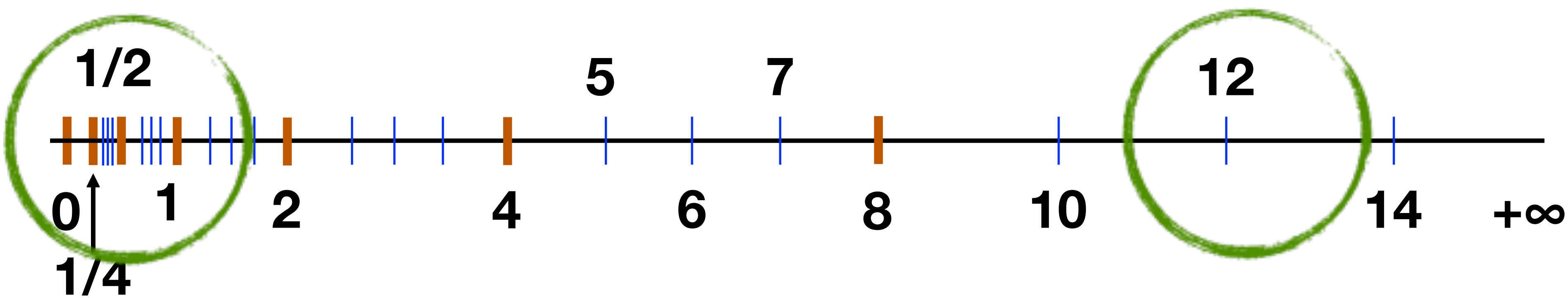


Representable Numbers (Positive Only)

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

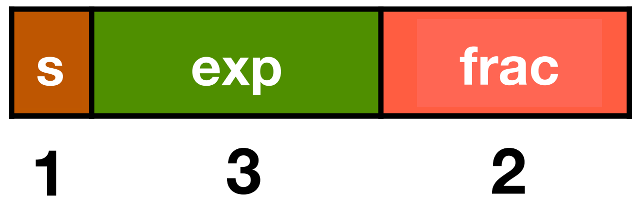
s	exp	frac
1	3	2

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



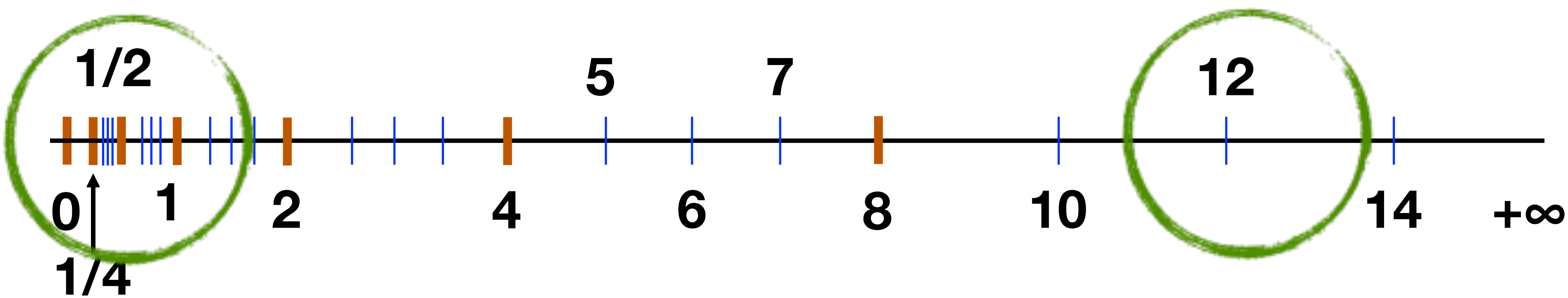
Representable Numbers (Positive Only)

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



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

- **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

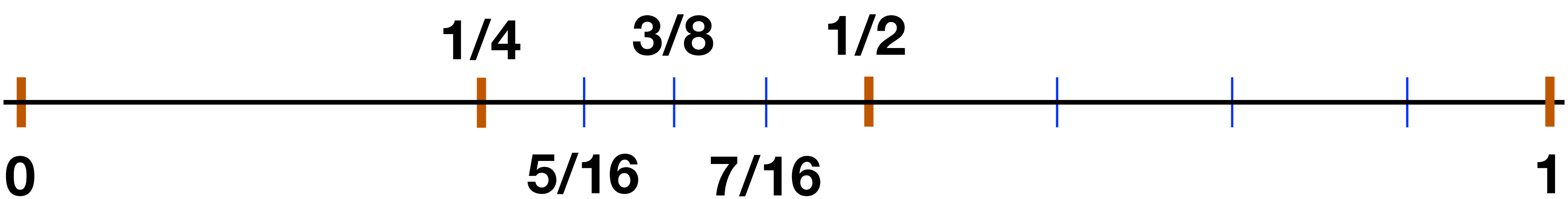


Representable Numbers (Positive Only)

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

s	exp	frac

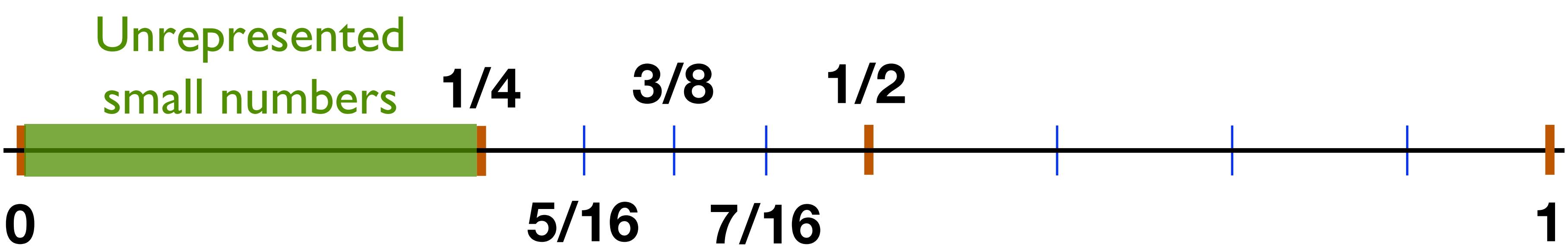
E	exp	E	exp
~~-3~~	~~000~~	1	100
-2	001	2	101
-1	010	3	110
0	011	~~4~~	~~111~~



Representable Numbers (Positive Only)

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

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

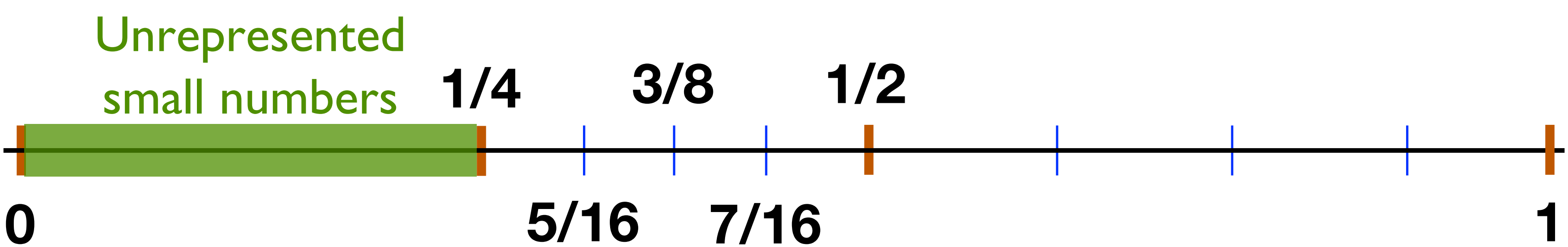


Representable Numbers (Positive Only)

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

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

- Always round to 0 is inelegant



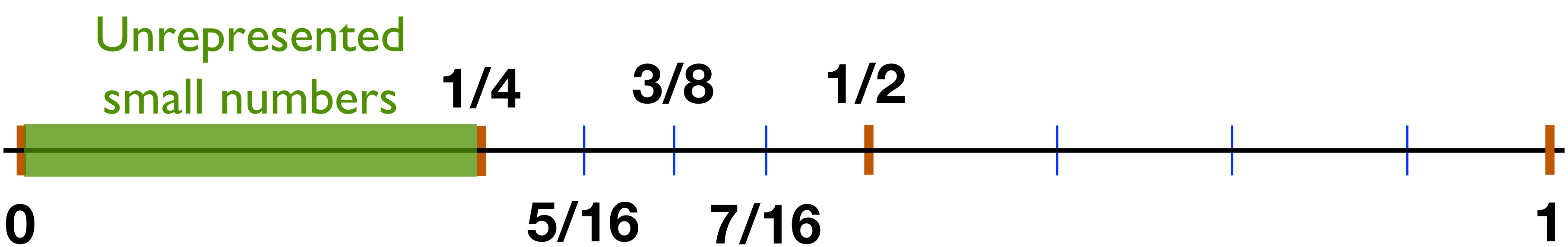
Representable Numbers (Positive Only)

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

s	exp	frac

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

- Always round to 0 is inelegant



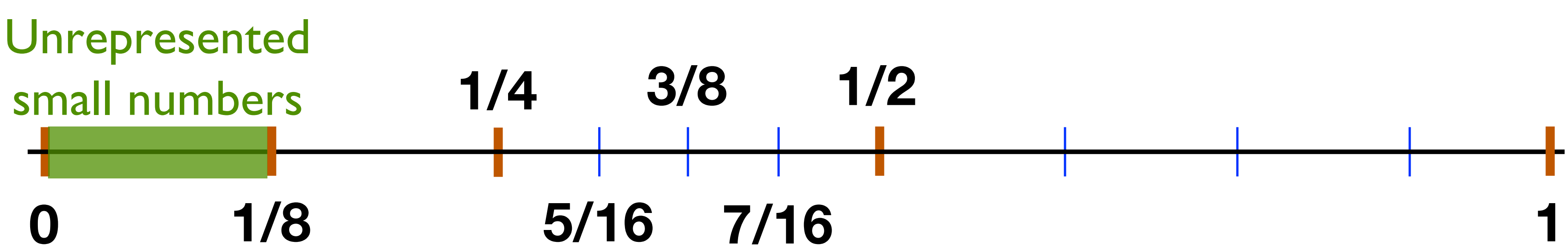
Representable Numbers (Positive Only)

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

s	exp	frac

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

- Always round to 0 is inelegant

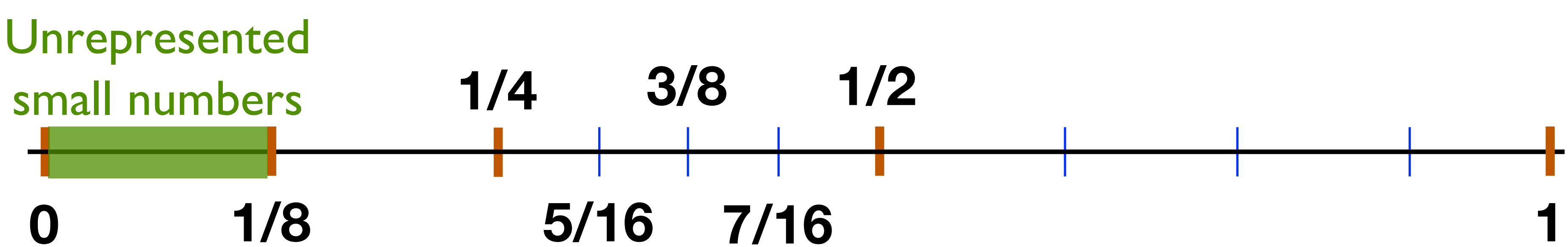


Representable Numbers (Positive Only)

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

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

- Always round to 0 is inelegant
- Using 000 for *exp* would only “delay” the problem rather than solving it



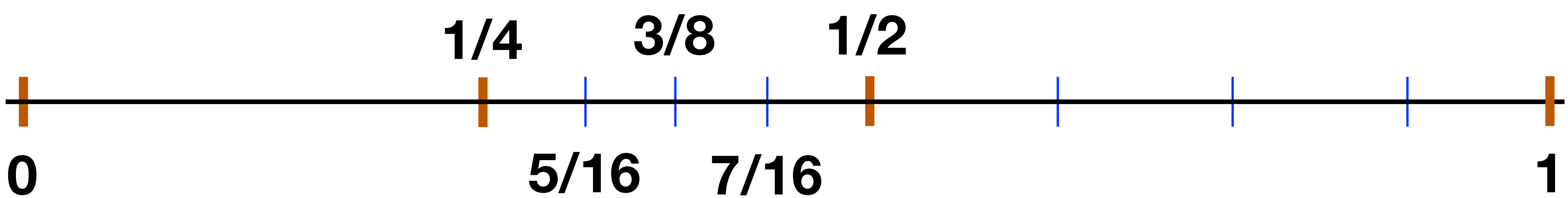
Subnormal (De-normalized) Numbers

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

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/denormalized numbers)

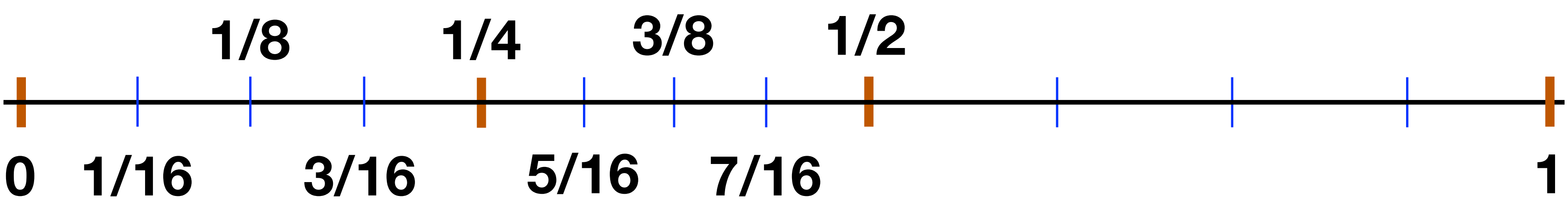


Subnormal (De-normalized) Numbers

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

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/denormalized numbers)



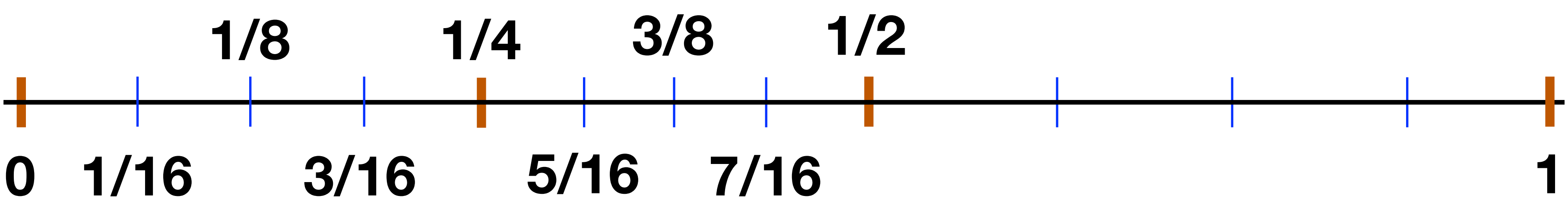
Subnormal (De-normalized) Numbers

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

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/denormalized numbers)
- *E* = (*exp* + 1) – bias (instead of *exp* - *bias*)
- *M* = 0.*frac* (instead of 1.frac)



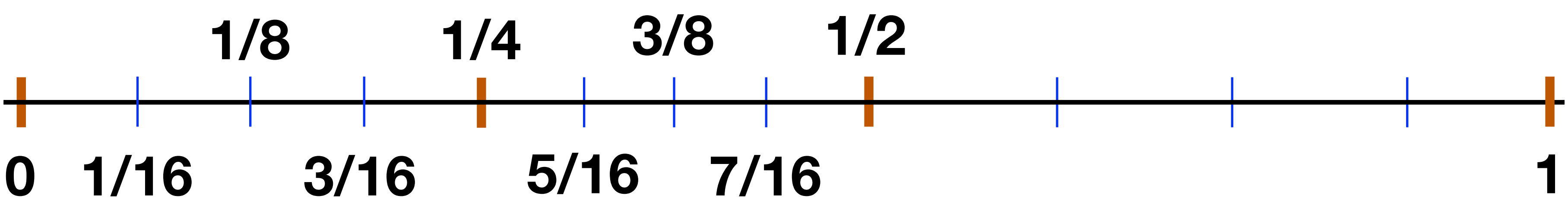
Subnormal (De-normalized) Numbers

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

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/denormalized numbers)
- E = (*exp* + 1) – bias (instead of *exp* - *bias*)
- M = 0.*frac* (instead of 1.frac)



0	000	01

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

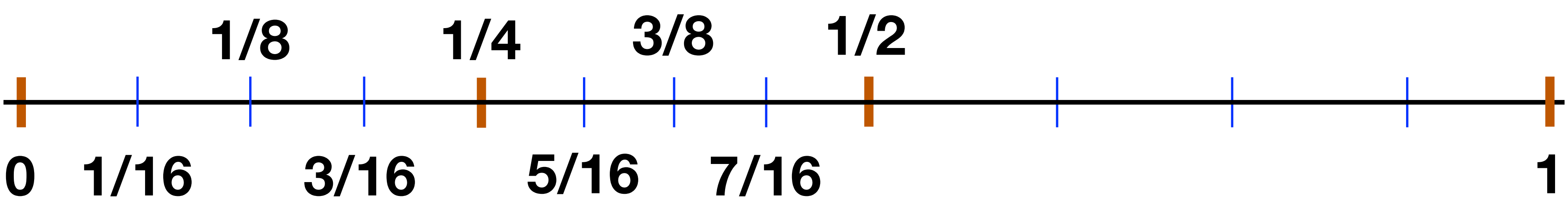
Subnormal (De-normalized) Numbers

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

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/denormalized numbers)
- *E* = (*exp* + 1) – 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

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

s	exp	frac

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

Special Values

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

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
  - +/- ∞, Not-a-Numbers (NaNs) (e.g., 0 / 0, 0 / ∞, ∞ / ∞, sqrt(–1), ∞ - ∞, ∞ x 0, etc.)

Special Values

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

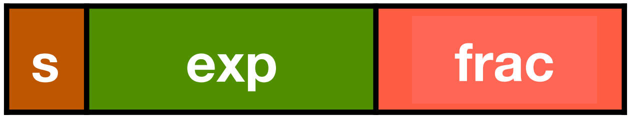
s	exp	frac

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
  - +/- ∞, Not-a-Numbers (NaNs) (e.g., 0 / 0, 0 / ∞, ∞ / ∞, sqrt(–1), ∞ - ∞, ∞ x 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
  - +/- ∞, Not-a-Numbers (NaNs) (e.g., 0 / 0, 0 / ∞, ∞ / ∞, sqrt(–1), ∞ - ∞, ∞ x 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
  - +/- ∞, Not-a-Numbers (NaNs) (e.g., 0 / 0, 0 / ∞, ∞ / ∞, sqrt(–1), ∞ - ∞, ∞ x 0, etc.)
- exp = 111 is reserved to represent these numbers
- exp = 111, frac = 00
  - +/- ∞ (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

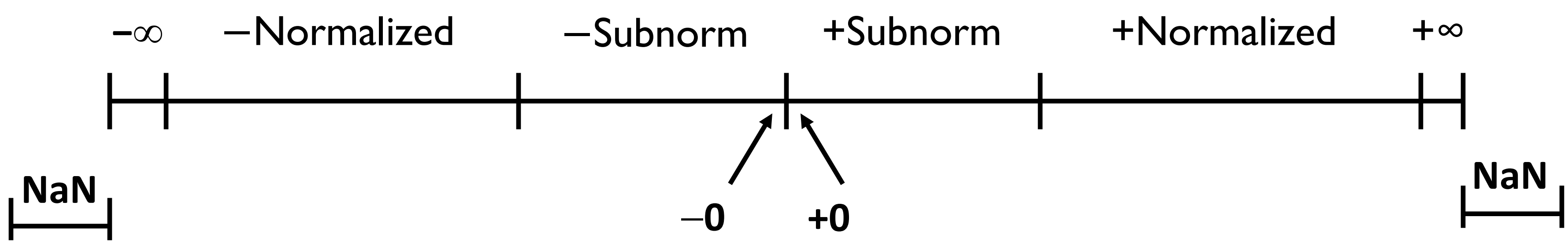
$v = (-1)^s M 2^E$

s	exp	frac

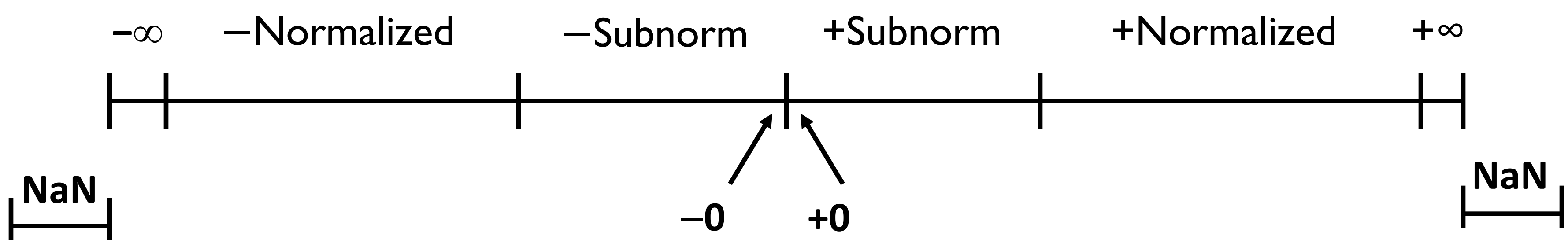
E	exp	E	exp
-2	000	1	100
-2	001	2	101
-1	010	3	110
0	011		111

- There are many special values in scientific computing
    - +/- ∞, Not-a-Numbers (NaNs) (e.g., 0 / 0, 0 / ∞, ∞ / ∞, sqrt(–1), ∞ - ∞, ∞ x 0, etc.)
- exp = 111 is reserved to represent these numbers
- exp = 111, frac = 00
    - +/- ∞ (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 != 00
    - Represent NaNs

Visualization: Floating Point Encodings

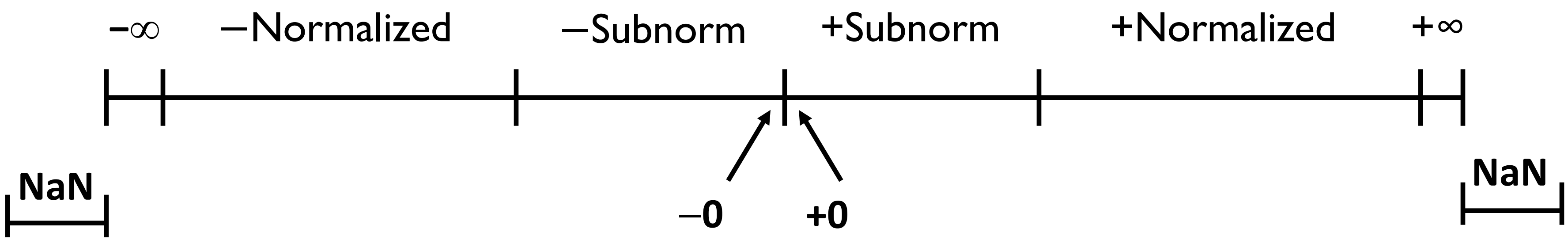


Visualization: Floating Point Encodings

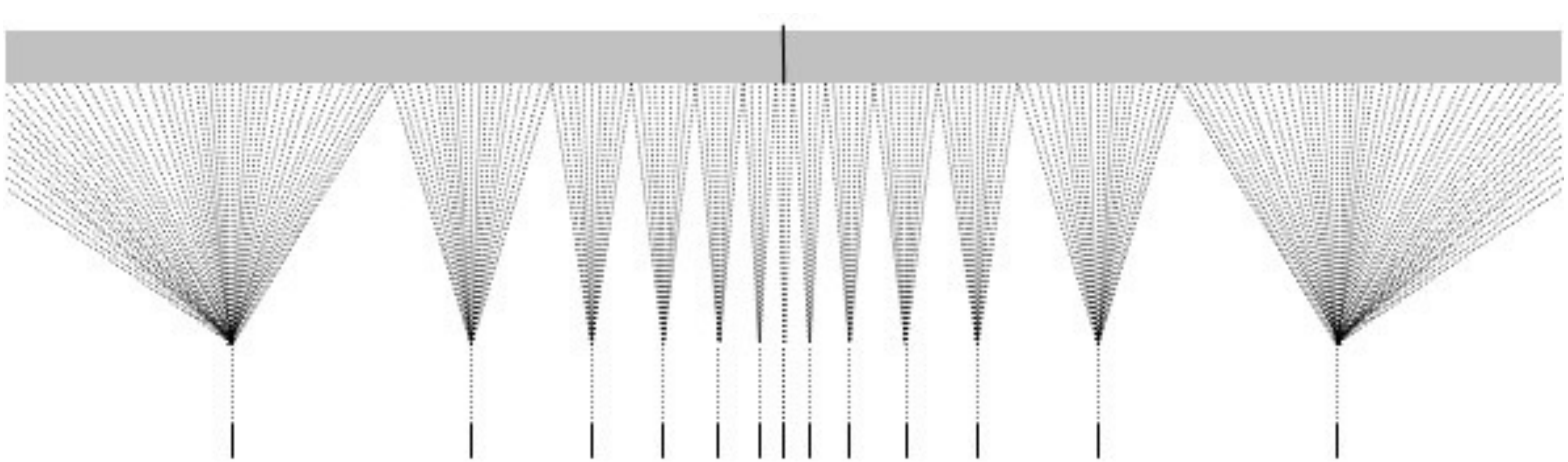


Infinite Amount of Real Numbers

Visualization: Floating Point Encodings

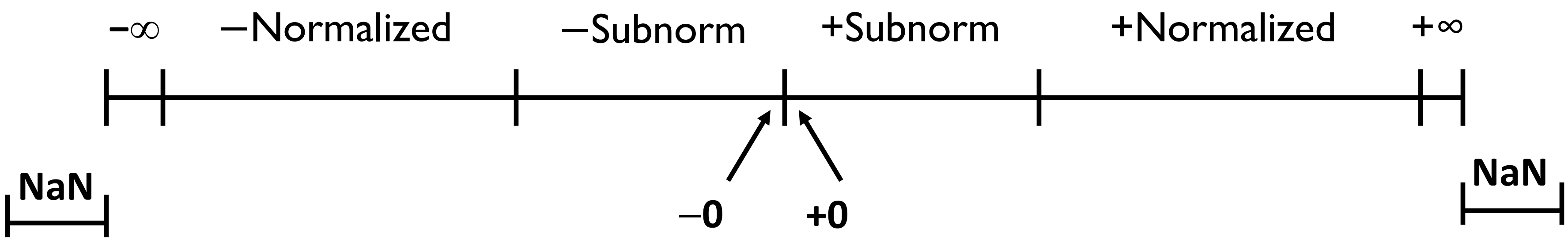


Infinite Amount of Real Numbers

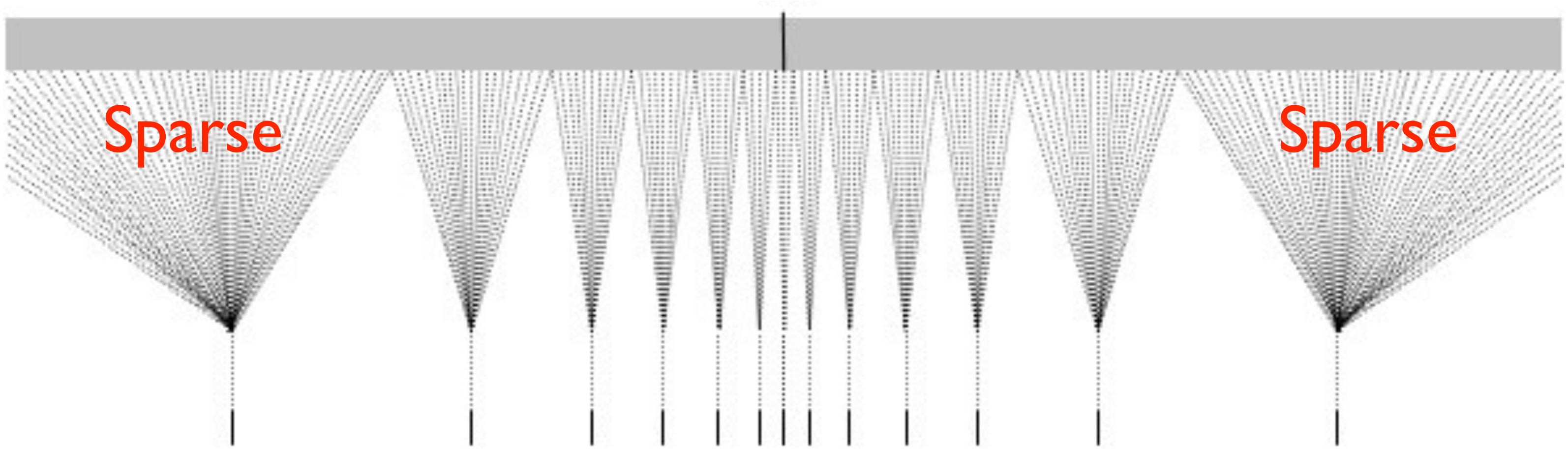


Finite Amount of Floating Point Numbers

Visualization: Floating Point Encodings

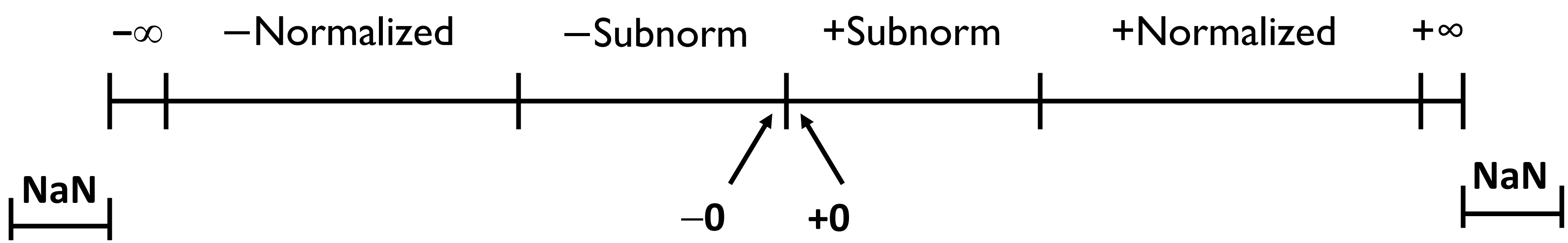


Infinite Amount of Real Numbers

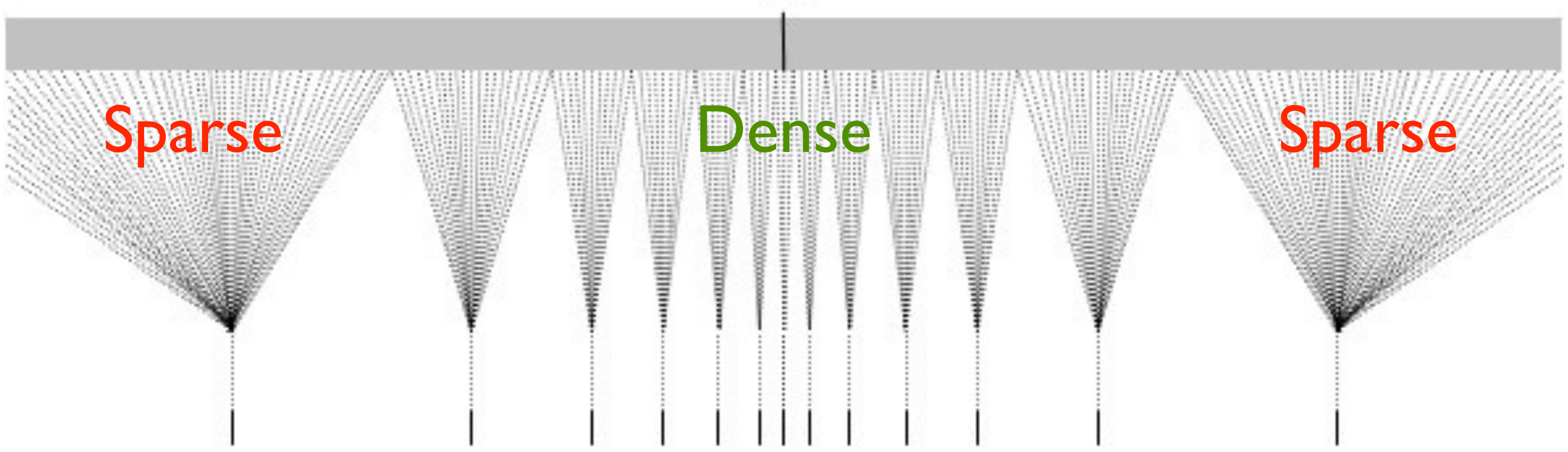


Finite Amount of Floating Point Numbers

Visualization: Floating Point Encodings



Infinite Amount of Real Numbers



Finite Amount of Floating Point Numbers

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 754 Floating Point Standard

- Single precision: 32 bits

- Double precision: 64 bits

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

Single Precision (32-bit) Example

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

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

s	exp	frac
1	8-bit	23-bit

Single Precision (32-bit) Example

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

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

s	exp	frac
1	8-bit	23-bit

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

Single Precision (32-bit) Example

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

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

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

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

0	exp	frac
1	8-bit	23-bit

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

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

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

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

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

Single Precision (32-bit) Example

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

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

0	10001100	11011011011010000000000
1	8-bit	23-bit

$$15213_{10} = 11101101101101_2$$
$$= (-1)^0 \, 1.\boxed{1101101101101}_2 \times 2^{\boxed{13}}$$

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

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 Computations

- The problem: Computing on floating point numbers might produce a result that can't be precisely represented
- 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

Rounding Modes (Decimal)

- Common ones:
  - Towards zero (chop)
  - Round down ($-\infty$)
  - Round up ($+\infty$)

Rounding Modes (Decimal)

- Common ones:
  - 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 (Decimal)

- Common ones:
  - Towards zero (chop)
  - Round down (-∞)
  - Round up (+∞)
- Nearest Even: Round to nearest; if equally near, then to the one having an even least significant digit (bit)

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)

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

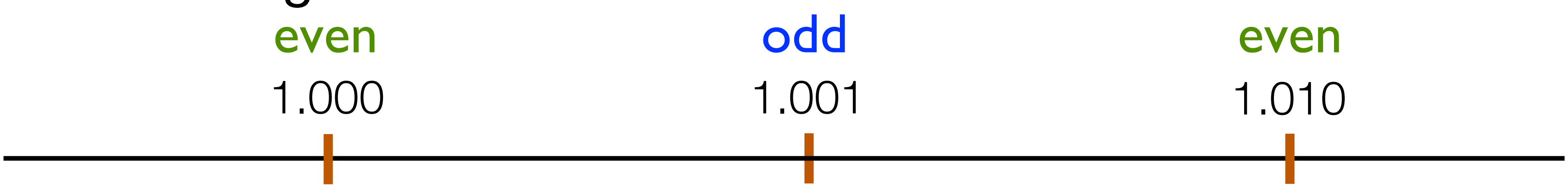
Rounding Modes (Binary Example)

- Nearest Even; 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)

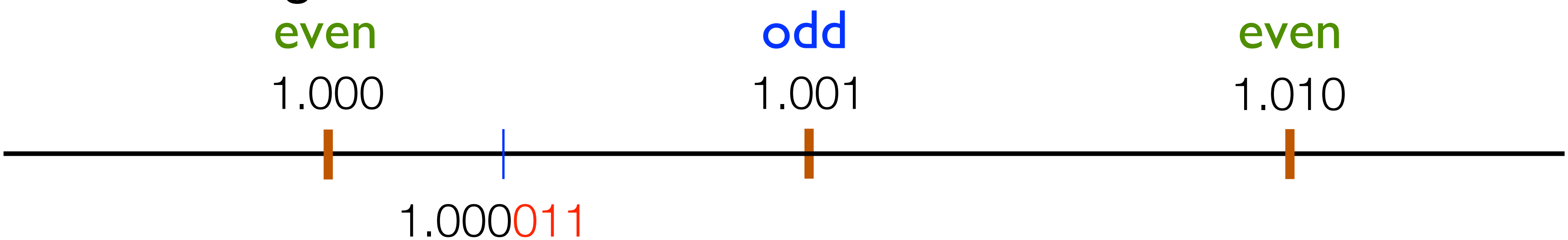
- Nearest Even; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



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)

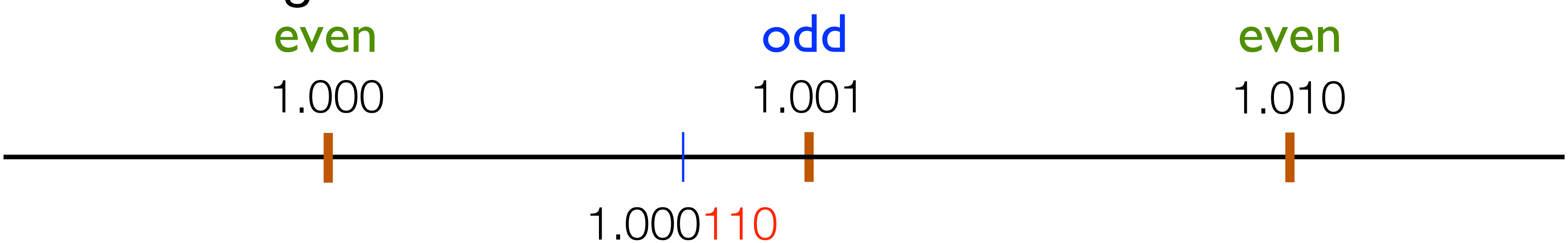
- Nearest Even; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



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)

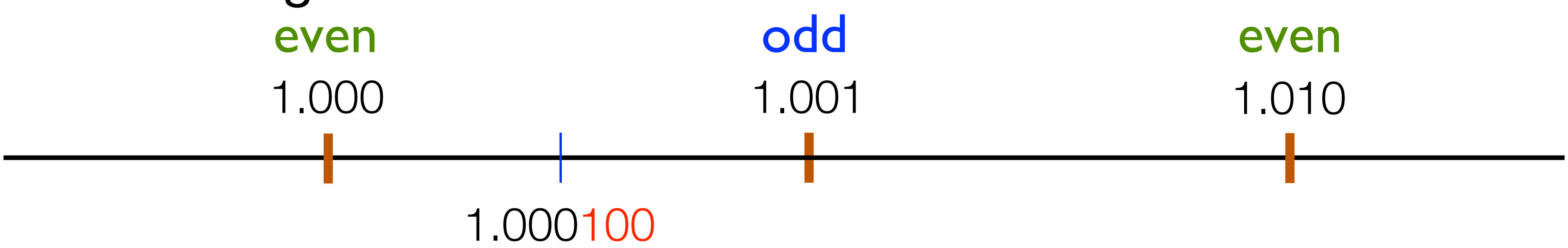
- Nearest Even; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



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)

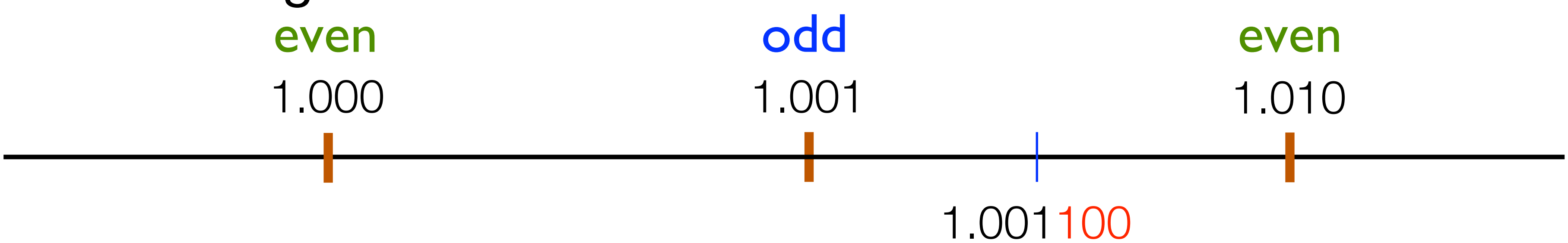
- Nearest Even; if equally near, then to the one having an even least significant digit (bit)
- Assuming 3 bits for *frac*



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)

- Nearest Even; 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

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$ $\qquad 1.000 \times 2^{-1} + 1.101 \times 2^{-3}$

Floating Point Addition

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

$1.000 \times 2^{-1} + 1.101 \times 2^{-3}$

align $1.000 \times 2^{-1} + 0.1101 \times 2^{-1}$

Floating Point Addition

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

$$1.000 \times 2^{-1} + 1.101 \times 2^{-3}$$

$$\downarrow$$

$$1.000 \times 2^{-1} + 0.1101 \times 2^{-1}$$

$$\downarrow$$

add $\quad 1.1101 \times 2^{-1}$

Floating Point Addition

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

$1.000 \times 2^{-1} + 1.101 \times 2^{-3}$

$\downarrow$

$1.000 \times 2^{-1} + 0.1101 \times 2^{-1}$

$\downarrow$

1.1101×2^{-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

$$1.000 \times 2^{-1} + 1.101 \times 2^{-3}$$

$$\downarrow$$

$$1.000 \times 2^{-1} + 0.1101 \times 2^{-1}$$

$$\downarrow$$

$$1.1101 \times 2^{-1}$$

Floating Point Addition

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

$$1.000 \times 2^{-1} + 1.101 \times 2^{-3}$$

$$1.000 \times 2^{-1} + 0.1101 \times 2^{-1}$$

$$1.1101 \times 2^{-1}$$

$$1.110 \times 2^{-1}$$

Mathematical Properties of FP Add

Mathematical Properties of FP Add

- Commutative? $a+b = b+a$

Mathematical Properties of FP Add

- Commutative? a+b =b+a ***Yes***

Mathematical Properties of FP Add

- Commutative? $a+b = b+a$ ***Yes***
- Associative? $a+b+c = a+(b+c)$

Mathematical Properties of FP Add

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

Mathematical Properties of FP Add

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

$$1.000 \times 2^{5} + 1.101 \times 2^{-3}$$

Mathematical Properties of FP Add

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

$$1.000 \times 2^{5} + 1.101 \times 2^{-3}$$

$$\downarrow$$

$$1.000 \times 2^{5} + 0.00000001101 \times 2^{5}$$

Mathematical Properties of FP Add

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

$$1.000 \times 2^5 + 1.101 \times 2^{-3}$$

$$\downarrow$$

$$1.000 \times 2^5 + 0.00000001101 \times 2^5$$

$$\downarrow$$

$$1.00000001101 \times 2^5$$

Mathematical Properties of FP Add

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

$$1.000 \times 2^{5} + 1.101 \times 2^{-3}$$

$$\downarrow$$

$$1.000 \times 2^{5} + 0.00000001101 \times 2^{5}$$

$$\downarrow$$

$$1.00000001101 \times 2^{5}$$

$$\downarrow$$

$$1.000 \times 2^{5}$$

Mathematical Properties of FP Add

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

Mathematical Properties of FP Add

- Commutative? a+b =b+a **Yes**
- Associative? a+b+c = a+(b+c) **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? a+b =b+a **Yes**
- Associative? a+b+c = a+(b+c) **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? a+b =b+a **_Yes_**
- Associative? a+b+c = a+(b+c) **_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? a+b =b+a **Yes**
- Associative? a+b+c = a+(b+c) **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? a+b =b+a **Yes**
- Associative? a+b+c = a+(b+c) **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? $a+b = b+a$ ***Yes***
- Associative? $a+b+c = a+(b+c)$ ***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

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

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*: $E1 + E2$

Floating Point Multiplication

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \times\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign *s*: $s1$ ^ $s2$
  - 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

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \times\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s: $s1$ ^ $s2$
  - 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
- 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?
  - a*(b+c) = a*b+a*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***
  - a*(b+c) = a*b+a*c?
  - 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**
  - a*(b+c) = a*b+a*c?
  - 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***
  - a*(b+c) = a*b+a*c?
  - 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

Floating Point in C

Category	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	$2^{63} - 1$	$\sim 9.2 \times 10^{18}$
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 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	$2^{63} - 1$	$\sim 9.2 \times 10^{18}$
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 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	$2^{63} - 1$	$\sim 9.2 \times 10^{18}$
SP floating point	`float`	32	$(2 - 2^{-23}) \times 2^{127}$	$\sim 3.4 \times 10^{38}$
DP floating point	`double`	64	$(2 - 2^{-52}) \times 2^{1023}$	$\sim 1.8 \times 10^{308}$

- To represent 2^{31} in fixed-point, you need at least 32 bits
  - Because fixed-point is a weighted positional representation
- In floating-point, we directly encode the exponent
  - Floating point is based on scientific notation
  - Encoding 31 only needs 6 bits in the *exp* field

Floating Point in C

- **`double/float`** $\rightarrow$ **`int`**
  - Truncates fractional part
  - Like rounding toward zero
  - Not defined when out of range or NaN

Floating Point in C

- **`double/float` → `int`**
  - Truncates fractional part
  - Like rounding toward zero
  - Not defined when out of range or NaN
- **`int` → `float`**
  - Can’t guarantee exact casting. Will round according to rounding mode

- **`int` → `double`**

Floating Point Review

$$v = (-1)^s \times 1.\mathit{frac} \times 2^E$$

s	exp	frac

Floating Point Review

$$v = (-1)^s \times 1.\mathit{frac} \times 2^E$$

Denormalized

s	*exp*	*frac*	Value	Value
0	000	00	0.00×2^{-2}	0
0	000	11	0.11×2^{-2}	3/16

- Denormalized (exp == 000)
  - E = (exp + 1) – bias
  - M = 0.frac

Floating Point Review

$$v = (-1)^s \times 1.frac \times 2^E$$

s	exp	frac

- Denormalized (exp == 000)
  - E = (exp + 1) – bias
  - M = 0.frac
- Normalized (exp != 000)
  - E = exp – bias
  - M = 1.frac

Denormalized

Normalized

s	exp	frac	Value	Value
0	000	00	0.00×2^{-2}	0
0	000	11	0.11×2^{-2}	3/16
0	001	00	1.00×2^{-2}	1/4
0	001	11	1.11×2^{-2}	7/16
0	010	00	1.00×2^{-1}	1/2
0	010	11	1.11×2^{-1}	7/8
0	100	00	1.00×2^{0}	1
0	100	11	1.11×2^{0}	1 3/4
0	101	00	1.00×2^{1}	2
0	101	11	1.11×2^{1}	3 1/2
0	110	00	1.00×2^{2}	4
0	110	11	1.11×2^{2}	7

Floating Point Review

$v = (-1)^s \times 1.\mathit{frac} \times 2^E$

- Denormalized (exp == 000)
  - E = (exp + 1) – bias
  - M = 0.frac
- Normalized (exp != 000)
  - E = exp – bias
  - M = 1.frac

	s	***exp***	***frac***	**Value**	**Value**
Denormalized	0	000	00	0.00×2^{-2}	0
	0	000	11	0.11×2^{-2}	3/16
Normalized	0	001	00	1.00×2^{-2}	1/4
	0	001	11	1.11×2^{-2}	7/16
	0	010	00	1.00×2^{-1}	1/2
	0	010	11	1.11×2^{-1}	7/8
	0	100	00	1.00×2^{0}	1
	0	100	11	1.11×2^{0}	1 3/4
	0	101	00	1.00×2^{1}	2
	0	101	11	1.11×2^{1}	3 1/2
	0	110	00	1.00×2^{2}	4
	0	110	11	1.11×2^{2}	7
Special Value	0	111	00	infinite	infinite
	0	111	11	NaN	NaN

Floating Point Review

		s	exp	frac	Value	Value
Denormalized		0	000	00	0.00×2^{-2}	0
		0	000	11	0.11×2^{-2}	3/16
Normalized		0	001	00	1.00×2^{-2}	1/4
		0	001	11	1.11×2^{-2}	7/16
		0	010	00	1.00×2^{-1}	1/2
		0	010	11	1.11×2^{-1}	7/8
		0	100	00	1.00×2^{0}	1
		0	100	11	1.11×2^{0}	1 3/4
		0	101	00	1.00×2^{1}	2
		0	101	11	1.11×2^{1}	3 1/2
		0	110	00	1.00×2^{2}	4
		0	110	11	1.11×2^{2}	7
Special Value		0	111	00	infinite	infinite
		0	111	11	NaN	NaN

Floating Point Review

- Bit patterns representing non-negative numbers are ordered the same way as integers, so could use regular integer comparison.

	s	exp	frac	Value	Value
Denormalized	0	000	00	0.00×2^{-2}	0
	0	000	11	0.11×2^{-2}	3/16
Normalized	0	001	00	1.00×2^{-2}	1/4
	0	001	11	1.11×2^{-2}	7/16
	0	010	00	1.00×2^{-1}	1/2
	0	010	11	1.11×2^{-1}	7/8
	0	100	00	1.00×2^{0}	1
	0	100	11	1.11×2^{0}	1 3/4
	0	101	00	1.00×2^{1}	2
	0	101	11	1.11×2^{1}	3 1/2
	0	110	00	1.00×2^{2}	4
	0	110	11	1.11×2^{2}	7
Special Value	0	111	00	infinite	infinite
	0	111	11	NaN	NaN

Floating Point Review

- Bit patterns representing non-negative numbers are ordered the same way as integers, so could use regular integer comparison.
- You don't get this property if:
  - *exp* is interpreted as signed
  - *exp* and *frac* are swapped

Denormalized

Normalized

Special Value

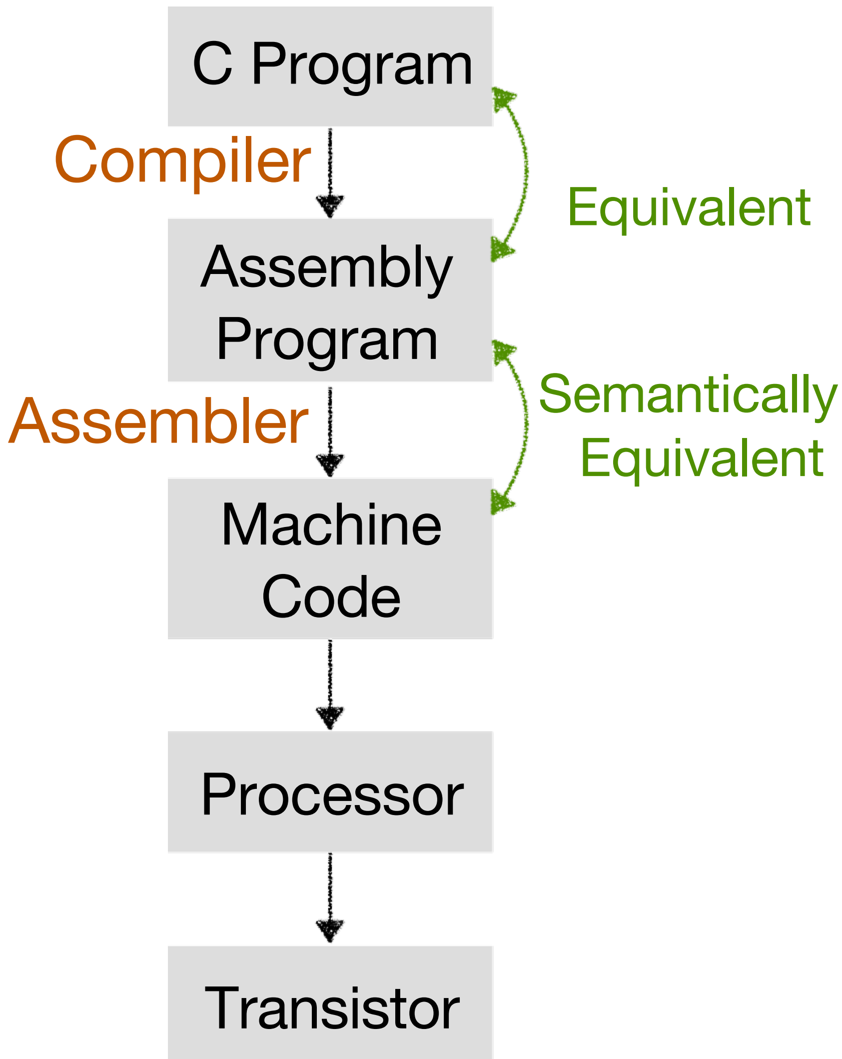
s	*exp*	*frac*	Value	Value
0	000	00	0.00×2^{-2}	0
0	000	11	0.11×2^{-2}	3/16
0	001	00	1.00×2^{-2}	1/4
0	001	11	1.11×2^{-2}	7/16
0	010	00	1.00×2^{-1}	1/2
0	010	11	1.11×2^{-1}	7/8
0	100	00	1.00×2^{0}	1
0	100	11	1.11×2^{0}	1 3/4
0	101	00	1.00×2^{1}	2
0	101	11	1.11×2^{1}	3 1/2
0	110	00	1.00×2^{2}	4
0	110	11	1.11×2^{2}	7
0	111	00	infinite	infinite
0	111	11	NaN	NaN

So far in 252…

C Program

`int`, `float`
`if`, `else`
`+`, `-`, `>>`

So far in 252…



`int, float`
`if, else`
`+, -, >>`

`ret, call`
`fadd`, **`add`**
`jmp, jne`

00001111
01010101
11110000

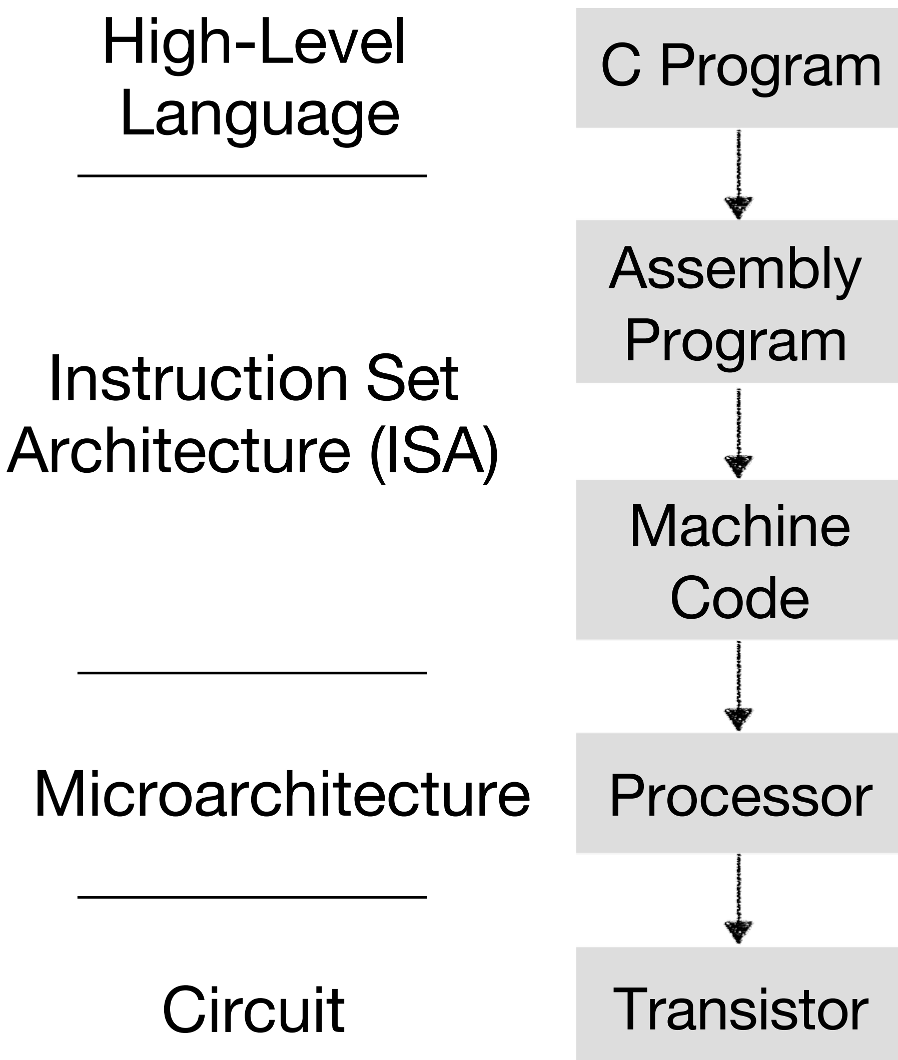
Fixed-point adder (e.g., ripple carry), Floating-point adder

NAND Gate
NOR Gate

So far in 252…

High-Level Language	C Program	**`int, float`** `if, else` `+, -, >>`
Instruction Set Architecture (ISA)	Assembly Program	`ret, call` **`fadd`**, **`add`** `jmp, jne`
	Machine Code	00001111 01010101 11110000
Microarchitecture	Processor	Fixed-point adder (e.g., ripple carry), Floating-point adder
Circuit	Transistor	NAND Gate NOR Gate

So far in 252…

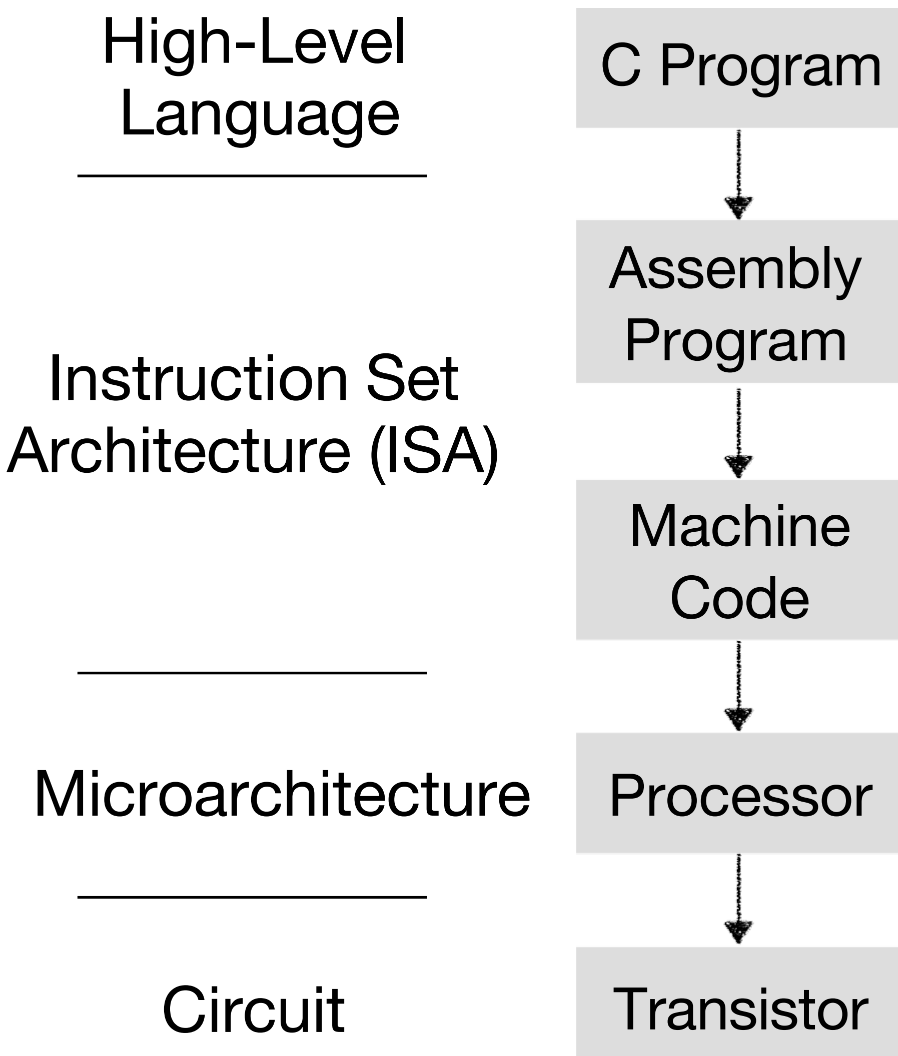

High-Level Language
C Program
Assembly Program
Instruction Set Architecture (ISA)
Machine Code
Microarchitecture
Processor
Circuit
Transistor

So far in 252…

High-Level Language	C Program
Instruction Set Architecture (ISA)	Assembly Program ↓ Machine Code
Microarchitecture	Processor
Circuit	Transistor

- ISA: Software programmers' view of a computer
  - Provide all info for someone wants to write assembly/machine code
  - "Contract" between assembly/ machine code and processor

So far in 252…



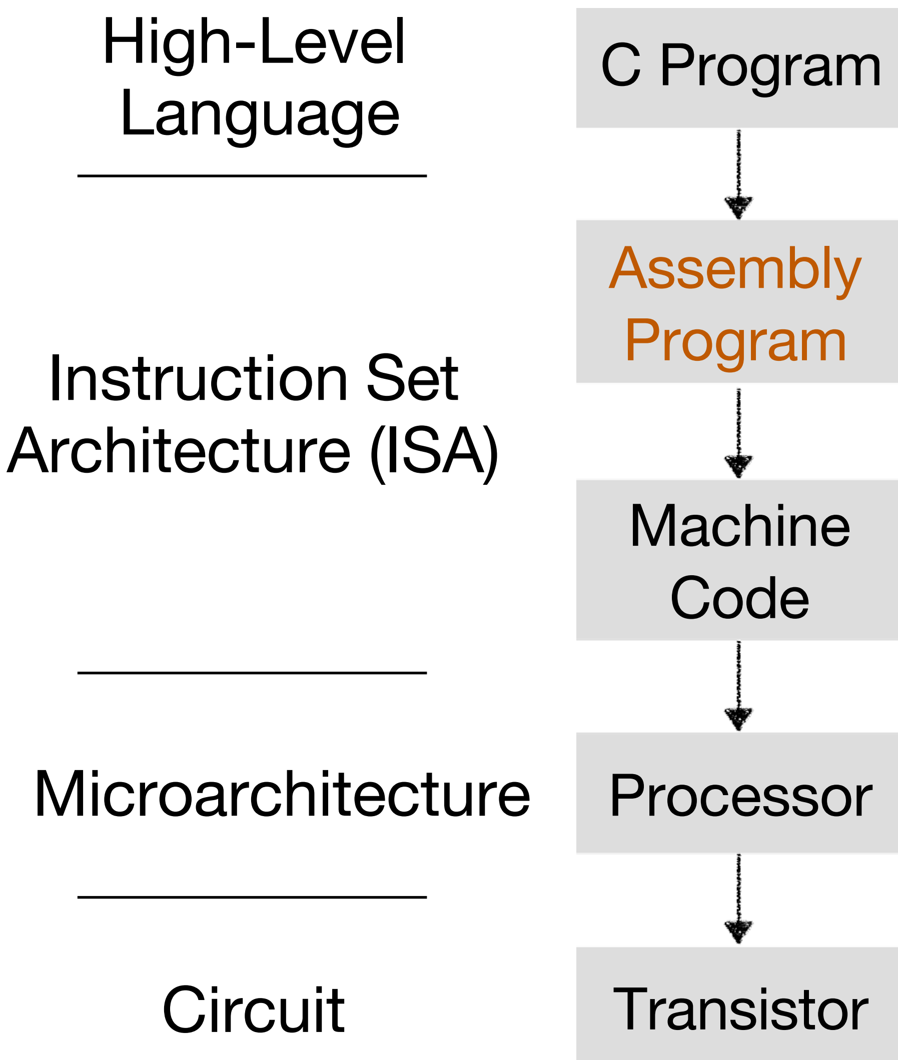
- ISA: Software programmers' view of a computer
  - Provide all info for someone wants to write assembly/machine code
  - "Contract" between assembly/ machine code and processor
- Processors execute machine code (binary). Assembly program is merely a text representation of machine code

So far in 252…

High-Level Language	C Program
Instruction Set Architecture (ISA)	Assembly Program
	Machine Code
Microarchitecture	Processor
Circuit	Transistor

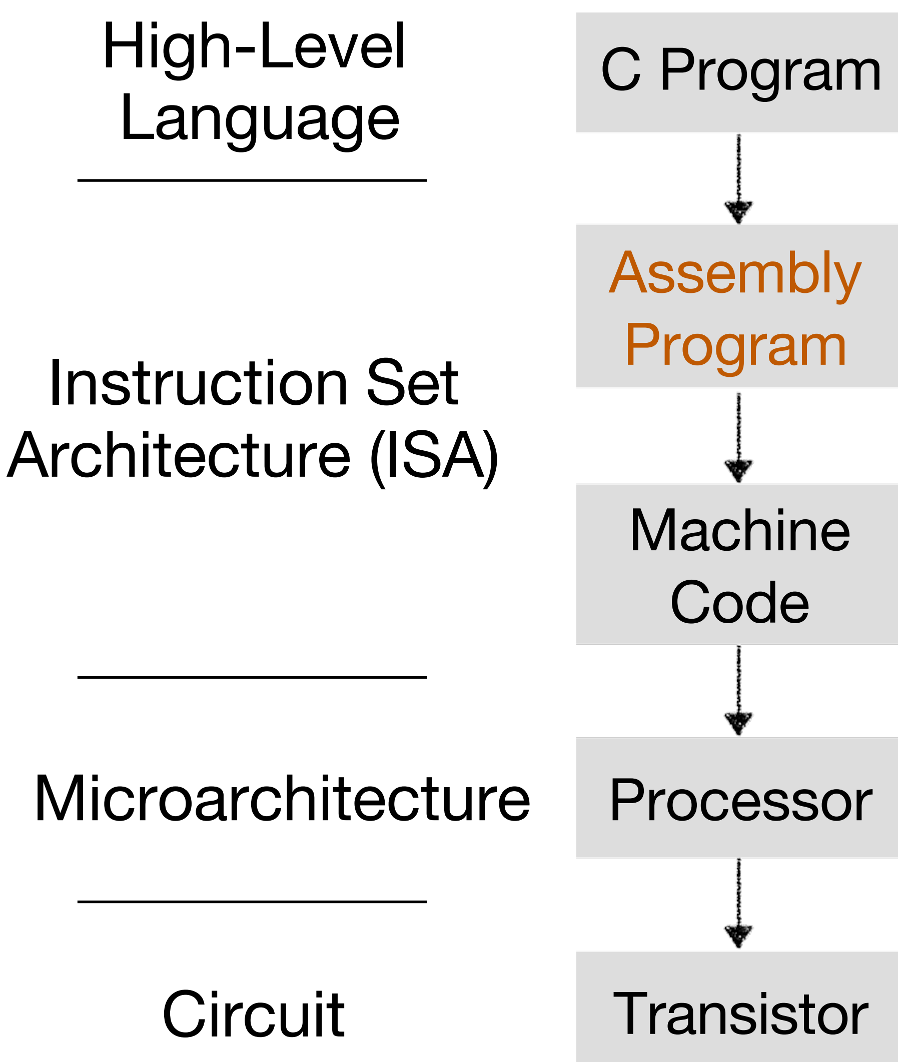
- ISA: Software programmers' view of a computer
  - Provide all info for someone wants to write assembly/machine code
  - "Contract" between assembly/ machine code and processor
- Processors execute machine code (binary). Assembly program is merely a text representation of machine code
- Microarchitecture: Hardware implementation of the ISA (with the help of circuit technologies)

This Module (4-5 Lectures)



- Assembly Programming
  - Explain how various C constructs are implemented in assembly code
  - Effectively translating from C to assembly program manually
  - Helps us understand how compilers work
  - Helps us understand how assemblers work

This Module (4-5 Lectures)



- Assembly Programming
  - Explain how various C constructs are implemented in assembly code
  - Effectively translating from C to assembly program manually
  - Helps us understand how compilers work
  - Helps us understand how assemblers work
- Microarchitecture is the topic of the next module