lecture 2

- fixed point

- IEEE floating point standard

Fixed point

Fixed point means we have a constant number of bits (or digits) to the left and right of the binary (or decimal) point.

Examples :

23953223.49 (base 10)

Currency uses a fixed number of digits to the right.

10.1101 (base 2)

Two's complement for fixed point numbers

e.g. 0110.1000 which is 6.5 in decimal

How do we represent -6.5 in fixed point ?

```
  0110.1000
  1001.0111    <----- invert bits
+ 0000.0001    <----- add  .0001
  0000.0000
```

Thus,

```
  1001.0111    <----- invert bits
+ 0000.0001    <----- add  .0001
  1001.1000    <-----  answer:  -6.5  in (signed) fixed point
```

Scientific Notation (floating point)

$$300{,}000{,}000 = 3 \times 10^{8}$$

$$= 3.0E+8$$

$$.000000456 = 4.56E-6$$

"Normalized" : one digit to the left of the decimal point.

Scientific Notation in binary

$$(1000.01)_2 = 1.00001 \times 2^3$$

$$(0.111)_2 = 1.11 \times 2^{-1}$$

"Normalized" means one "1" bit to the left of the binary point. **(Note that 0 cannot be represented this way.)**

sign "exponent"

$$\pm \; 1.\_\_\_\_\_\_\_\_ \times 2^{E}$$

"significand"

(also called "mantissa")

How to represent this information ?

How to represent the number 0 ?

IEEE floating point standard (est. 1985)

case 1: single precision (32 bits = 4 bytes)

1	8	23
sign	"exponent"	"significand"

31, 30 . . 23, 22, . . . , 1, 0

Let's look at these three parts, and then examples.

sign 0 for positive, 1 for negative

"significand"

You don't encode the "1" to the left of the binary point.

Only encode the first 23 bits to the right of the binary point.

exponent code	exponent value
00000000	**reserved (explained soon)**
00000001	-126
00000010	-125
00000011	- 124
:	:
:	:
01111111	**0**
10000000	1
10000001	2
:	:
:	:
11111110	127
11111111	**reserved (explained soon)**

This is not two's complement !

unsigned exponent code = exponent value + "bias"
(for 8 bits, bias is defined to be 127)

Q: What is the largest positive normalized number ?

(single precision)

$$1.\_\_\_\_\_\_\_\_\_\_\_\_ \times 2^{e}$$

A:

$$1.1111\ldots\ldots11 \times 2^{127}$$

$$2^{127} \approx 10^{?}$$

$$2^{10} \approx 10^{3}$$

$$\begin{aligned} 2^{127} &= 2^{120} \cdot 2^{7} \\ &= (2^{10})^{12} \cdot 2^{7} \\ &\approx (10^{3})^{12} \cdot 10^{2} \\ &= 10^{38} \end{aligned}$$

Q: What is the smallest positive normalized number ?
(single precision)

$1.\underline{\hspace{6cm}} \times 2^{e}$

A:

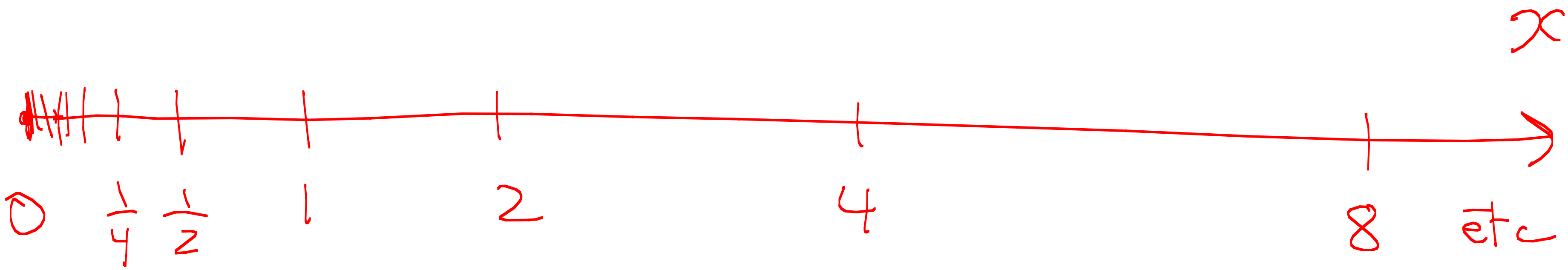
$1.000000\ldots0 \times 2^{-126}$

Exponent code 00000000 reserved for "denormalized" numbers

$$\pm\, 0.\,\underline{\hspace{10cm}} \times 2^{-126}$$

- belong to $(-2^{-126},\ 2^{-126})$
- includes 0

Dividing each power of 2 interval into 2^23 equal parts (same for negative real numbers).



Note the power of 2 intervals themselves are equally spaced on a log scale.

Exponent code 11111111 also reserved.

if significand is all 0's

then value is +- infinity (depending on sign bit)

else value is NaN ("not a number")
e.g. variable is declared but hasn't been assigned a value

Example: write 8.75 a single precision float (IEEE).

First convert to binary.

$$8.75$$

$$= (1000)_2 . (75)_{10}$$

$$= (10001)_2 . (5)_{10} \times 2^{-1}$$

$$= 100011.0 \times 2^{-2}$$

$$= 1.00011 \times 2^3$$

$(8.75)_{10} = (1.00011)_2$ x 2^3

23 bit significand: <u>00011</u>000000000000000000

exponent value: e = 3

exponent code = exponent value (e) + bias

Thus, exponent code is unsigned 3 + 127.

$(130)_{10} = (10000010)_2$

So, the 32 bit representation is :

0 10000010 00011000000000000000000

0 x 4 1 0 c 0 0 0 0

Recall last lecture: 0.05 cannot be represented exactly.

```
float x = 0;
for (int ct = 0; ct < 20; ct ++) {
  x += 1.0 / 20;
  System.out.println( x );
}
```

```
0.05
0.1
0.15
0.2
0.25
0.3
0.35000002
0.40000004
0.45000005
0.50000006
  etc
```

Floating Point Addition

$$x = 1.00100100010000010100001 * 2^2$$

$$y = 1.10101000000000000101010 * 2^{-3}$$

$$x + y = ?$$

Floating Point Addition

x = 1.00100100010000010100001 * 2^2

y = 1.10101000000000000101010 * 2^ {-3}

x + y = ?

x = 1.0010010001000001010000100000 * 2^2

y = .0000110101000000000000101010 * 2^2

but the result x+y has more than 23 bits of significand

How many *digits* (base 10) of precision can we represent with 23 *bits* (base 2) ?

$$2^{23}$$

$$= \left(2^{10}\right)^{2} 2^{3} \quad = (1024)^2 . 2^3$$

$$\approx 10^{6} \cdot 10^{1}$$

$$= 10^{7}$$

case 2: double precision (64 bits = 8 bytes)

1	11	52
sign	"exponent"	"significand"

63, 62 . . 52, 51, . . . , 1, 0

<u>exponent code</u>	<u>exponent value</u>

unsigned exponent code = exponent value + bias

For 11 bits, bias is defined to be 2^10 - 1 = 1023.

00000000000	**reserved**
00000000001	-1022
00000000010	-1021
00000000011	- 1020
:	:
:	:
01111111111	**0**
10000000000	1
10000000001	2
:	:
:	:
11111111110	1023
11111111111	**reserved**

Example

$(8.75)_{10} = (1.00011)_2 \times 2^3$

significand (52 bits)

= .0001100000000000000000000000000000....

exponent = 3, code using 11 bits:

$3 + 1023 = 1026 = (10000000010)_2$

double precision float (64 bits)

0 10000000010 0001100000000000000000000000000000...

0 x 4 0 2 1 8 0 0 0 0 0 000000

Q: What is the largest positive normalized number ?
(double precision)

$1. \underline{\hspace{6cm}} \times 2^{e}$

A:

$$2^{1023}$$

$$= \left(2^{10}\right)^{102} 2^{3}$$

$$\approx \left(10^{3}\right)^{102} 10$$

$$= 10^{307}$$

Approximation Errors (Java/C/...)

```
double x = 0;
for (int ct=0; ct < 10; ct ++) {
  x += 1.0 / 10;
  System.out.println( x );
}
```

```
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
```

How many *digits* of precision can we represent with 52 *bits* ?

$$2^{52}$$

$$= (2^{10})^5 \, 2^2$$

$$\approx (10^3)^5 \, 10$$

$$= 10^{16}$$

52 bits covers about the same "range" as 16 digits.

That is why the print out on the previous slide had up to (about) 16 digits to the right of the decimal point.