

lecture 2

floating point

↙ "decimal point"

$$26.375 = 2 \times 10^1 + 6 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}$$

↙ "binary point"

$$\begin{aligned} (11010.011)_{\text{two}} &= 2^4 + 2^3 + 2^1 + 2^{-2} + 2^{-3} \\ &= 16 + 8 + 2 + 0.25 + 0.125 \\ &= 26.375 \end{aligned}$$

Multiplying by 2 means shifting bits to left
i.e. shifting binary point to right.

$$\begin{aligned} &(11010.011)_{\text{two}} \times 2 \\ &= (2^4 + 2^3 + 2^1 + 2^{-2} + 2^{-3}) \times 2 \\ &= (110100.11)_{\text{two}} \end{aligned}$$

Dividing by 2 and not ignoring remainder means shifting bits to right,
i.e. shifting binary point to left.

$$\begin{aligned} &(11010.011)_{\text{two}} / 2 \\ &= (2^4 + 2^3 + 2^1 + 2^{-2} + 2^{-3}) / 2 \\ &= (1101.0011)_{\text{two}} \end{aligned}$$

Converting fractional numbers from binary to decimal is straight forward.

n	2^{-n}
-1	.5
-2	.25
-3	.125
-4	.0625
-5	.03125
-6	.015625
-7	.0078125
:	etc.

How to convert decimal to binary?

$$26.375 = (\underline{\quad?} \cdot \underline{\quad?})_{\text{two}}$$

m	b_i
26	
13	0
6	1
3	0
1	1
0	1

$\Rightarrow 26 = (11010)_{\text{two}}$

$$\begin{aligned}
 &.375 \\
 &= .75 \times 2^{-1} \\
 &= 1.5 \times 2^{-2} \\
 &= 3.0 \times 2^{-3} \\
 &= (11)_{\text{two}} \times 2^{-3} \\
 &= (.011)_{\text{two}}
 \end{aligned}$$

$$19.243 = (?)_{\text{two}}$$

<u>m</u>	<u>b_i</u>
19	
9	1
4	1
2	0
1	0
0	1

∴ $19 = (10011)_{\text{two}}$

$$19.243 = (10011. \text{---})_{\text{two}}$$

$$\begin{aligned}
 .243 &= (0)_{\text{two}} \cdot 486 \times 2^{-1} \\
 &= (00)_{\text{two}} \cdot 972 \times 2^{-2} \\
 &= (001)_{\text{two}} \cdot 944 \times 2^{-3} \\
 &= (0011)_{\text{two}} \cdot 888 \times 2^{-4} \\
 &= (00111)_{\text{two}} \cdot 776 \times 2^{-5} \\
 &\dots
 \end{aligned}$$

Thus $(.243)_{\text{ten}} = (.00111)_{\text{two}} + \sum_{i=6}^{\infty} b_i 2^{-i}$

$$19.243 = (10011.00111\text{---})_{\text{two}}$$

Note: We cannot get an **exact** representation using a **finite** number of bits for this example.

When we convert a decimal number **with a finite number of digits** into binary, we get:

- a finite number non-zero bits to left of binary point
 - an infinitely repeating sequence of bits to right of binary point
- eg. $.011 \underline{10110} \underline{10110} \underline{10110} \dots$

Why?

$$\begin{aligned}
 (.05)_{\text{ten}} &= (0)_{\text{two}} \cdot 1 \times 2^{-1} \\
 &= (00)_{\text{two}} \cdot 2 \times 2^{-2} \\
 &= (000)_{\text{two}} \cdot 4 \times 2^{-3} \\
 &= (0000)_{\text{two}} \cdot 8 \times 2^{-4} \\
 &= (00001)_{\text{two}} \cdot 6 \times 2^{-5} \\
 &= (000011)_{\text{two}} \cdot 2 \times 2^{-6} \\
 &= .00 \underline{0011} \underline{0011} \underline{0011} \dots
 \end{aligned}$$

this will repeat ad infinitum

Sign 0 - positive
 1 - negative

Significand

1. _____

You don't encode first "1".
 The 23 bits to right are sometimes called the fractional part.

Exponent

0000 0000 } reserved
 1111 1111 }

For the 254 other 8 bit codes
 • treat 8 bits as unsigned
 • subtract 127 ("bias") to give the exponent value.

<u>exponent code</u>	<u>exponent value</u>
0000 0000	reserved
0000 0001	-126
0000 0010	-125
⋮	⋮
0111 1111	0
1000 0000	1
1000 0001	2
⋮	⋮
1111 1110	127
1111 1111	reserved

What is the largest positive normalized number?

1. _____ $\times 2^e$

Answer:

1.1111 11 $\times 2^{127}$

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

What is the smallest positive normalized number?

1. _____ $\times 2^e$

Answer:

1.000000 0 $\times 2^{-126}$

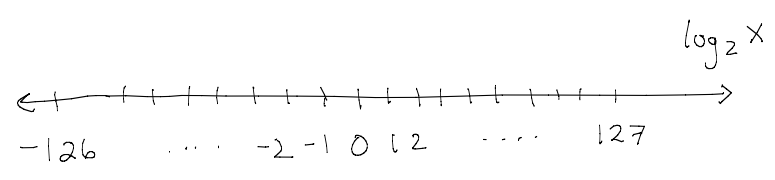
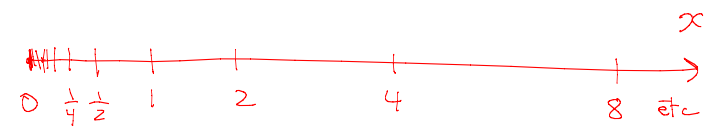
Exponent code 00000000

"denormalized" numbers

$\pm 0.\text{-----} \times 2^{-126}$

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

divide each interval into 2^{23} equal parts



Exponent code 11111111

if significant is all 0's

then value is $\pm \infty$
depending on sign bit

else value is "not a number" (NaN)

e.g. variable declared but not yet assigned a value

Example

Write 8.75 as single precision float.

$$\begin{aligned}
 &8.75 \\
 &= (1000)_{\text{two}} \cdot (75)_{\text{ten}} \\
 &= (10001) \cdot 5 \times 2^{-1} \\
 &= 100011.0 \times 2^{-2} \\
 &= 1.00011 \times 2^3
 \end{aligned}
 \left. \begin{array}{l} \\ \\ \\ \\ \end{array} \right\} \begin{array}{l} \text{significant} \\ e = 3 \\ \text{code using bias} \\ e + 127 \\ = 130 \\ = (10000010)_{\text{two}} \end{array}$$

$$\Rightarrow \underline{0 \ 1000010 \ 00011 \dots \ 0} \\
 = 0x410c0000$$

Approximation errors : Java Example

```

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

- 0.1
- 0.2
- 0.3
- 0.4
- 0.5
- 0.6
- 0.70000005
- 0.8000001
- 0.9000001
- 1.0000001

} 0's to the right.
How many?

$$2^{23}$$

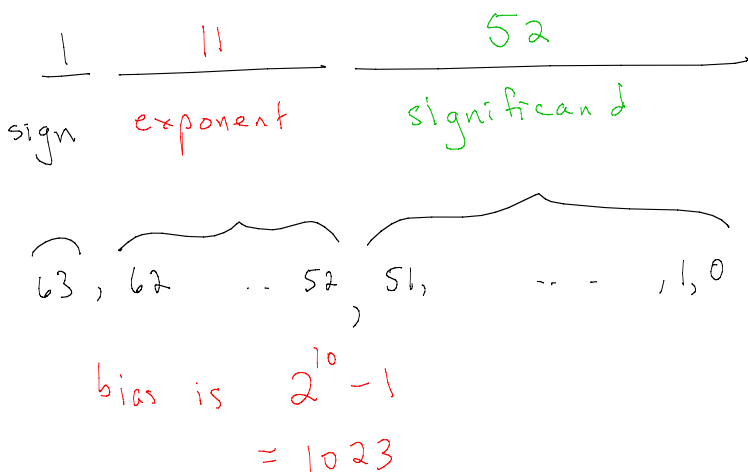
$$= (2^{10})^2 \cdot 2^3$$

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

$$= 10^7$$

23 bits of precision corresponds to about 7 (base 10) digits.

case 2: double precision



exponent code

000000000000
 000000000001
 000000000010
 ...
 011111111111
 100000000000
 100000000001
 ...
 111111111110
 111111111111

exponent value

reserved
 -1022
 -1021
 ...
 0
 1
 2
 ...
 1023
 reserved

$$\begin{aligned}
 & 2^{1023} \\
 &= (2^{10})^{102} \cdot 2^3 \\
 &\approx (10^3)^{102} \cdot 10 \\
 &= 10^{307}
 \end{aligned}$$

Example

Write 8.75 as double precision and express using hexadecimal.

$$8.75 = 1.0011 \times 2^3$$

$$\Rightarrow \left\{ \begin{array}{l} \text{.0011} \dots \text{ significand} \\ e = 3 \text{ code using } e + 1023 \\ \quad \quad \quad = 1026 \\ \quad \quad \quad = (1000000010)_{\text{two}} \end{array} \right.$$

$$\Rightarrow \underline{0 \ 1000 \ 0000 \ 0010 \ 0001 \ 1000} \\
 = 0x4021800000000000$$

Approximation errors : Java Example

```

double x = 0;
for (int i=0; i<10; i++) {
    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

```

$$\begin{aligned}
 & 2^{52} \\
 &= (2^{10})^5 \cdot 2^2 \\
 &\approx (10^3)^5 \cdot 10 \\
 &= 10^{16}
 \end{aligned}$$

52 bits of precision is about 16 digits

Computer arithmetic with floating point

$$\left. \begin{aligned} (a+b)+c &= a+(b+c) \\ (a*b)*c &= a*(b*c) \\ a*(b+c) &= a*b + a*c \end{aligned} \right\} \text{not necessarily}$$

Calculus:

$$\frac{df(x)}{dx} \approx \frac{f(x+\delta) - f(x)}{\delta} \text{ for small } \delta$$

This can produce weird stuff! (COMP 350 Numerical Computing)

- Exercises ↓ posted
- 273 prerequisites

