lecture 2

- fixed point
- IEEE floating point standard


For those interested in finding out what research is all about, I encourage you to participate in studies such as these.

Participants needed for Social Psychology research

* If you think you might like to be a paid participant in a study, you may enter the participation pool.
* Participants for this study will be contacted for an appointment and provided with instructions for arrival.
* Compensation is $15.00, depending on the study.
* All studies are approved by the Institutional Review Board.
* There is no obligation to participate in a study once you are in the pool, and you will not be contacted for an appointment at any time.

http://fluidsurveys.com/s/paidpool/

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
1001.1000 <----- invert bits
0000.0001 <----- add .0001
0000.0000

Thus,

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

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)

Scientific Notation (floating point)

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

Example:

300,000,000,000 = 3 \times 10^8

0.0000456 = 4.56 \times 10^{-6}

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

IEEE floating point standard (est. 1985)

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

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.

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

sign 0 for positive, 1 for negative

"significand"

(also called "mantissa")

How to represent this information?

How to represent the number 0?

Scientific Notation in binary

\[
(1.011)_2 = 1.011_2 \times 2^3
\]

\[
(0.111)_2 = 1.11 \times 2^{-1}
\]
### Exponent Code

<table>
<thead>
<tr>
<th>Exponent Code</th>
<th>Exponent Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>reserved (explained soon)</td>
</tr>
<tr>
<td>00000001</td>
<td>-126</td>
</tr>
<tr>
<td>00000010</td>
<td>-125</td>
</tr>
<tr>
<td>00000011</td>
<td>-124</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>01111111</td>
<td>0</td>
</tr>
<tr>
<td>10000000</td>
<td>1</td>
</tr>
<tr>
<td>10000001</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11111110</td>
<td>127</td>
</tr>
<tr>
<td>11111111</td>
<td>reserved (explained soon)</td>
</tr>
</tbody>
</table>

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

### Questions

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

**A:**

\[
\text{exponent code} \quad \text{exponent value} \\
00000000 \quad \text{reserved (explained soon)} \\
00000001 \quad -126 \\
00000010 \quad -125 \\
00000011 \quad -124 \\
\vdots \quad \vdots \\
01111111 \quad 0 \quad \text{This is not two's complement!} \\
10000000 \quad 1 \\
10000001 \quad 2 \\
\vdots \quad \vdots \\
11111110 \quad 127 \\
11111111 \quad \text{reserved (explained soon)}
\]

### Example

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

First convert to binary.

\[
8.75 = (1.00011)_2 \times 2^3
\]

Exponent code 00000000 reserved for "denormalized" numbers
- belong to (-2\^127, 2\^128)
- 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.

\[
(8.75)_{10} = (1.00011)_{2} \times 2^3
\]

23 bit significand: 000110000000000000000000
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 000110000000000000000000
Recall last lecture: 0.05 cannot be represented exactly.

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

<table>
<thead>
<tr>
<th>0.05</th>
<th>0.1</th>
<th>0.15</th>
<th>0.2</th>
<th>0.25</th>
<th>0.3</th>
<th>0.35000002</th>
<th>0.40000004</th>
<th>0.45000005</th>
<th>0.50000006</th>
</tr>
</thead>
</table>

Floating Point Addition

```java
x = 1.00100100010000010100001 * 2^2
y = 1.101010000000000101010 * 2^(-3)
```

```plaintext
x + y = ?
```

Floating Point Addition

```java
x = 1.00100100010000010100001 * 2^2
y = 1.101010000000000101010 * 2^(-3)
```

```plaintext
x + y = ?
```

```
x = 1.00100100010000010100001 * 2^2
y = .000110101000000000000101010 * 2^(-3)
```

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)?

```latex
\begin{align*}
2^{13} & = (2^{10})^2 \cdot 2^3 \\
\approx & \quad |0\cdot1|^{10} \\
= & \quad |1|^{0}\cdot1
\end{align*}
```

Floating Point Representation:

- **Sign**: 1 bit
- **Exponent**: 11 bits (normalized)
- **Significand**: 23 bits

```
\text{case 2: double precision} \quad (64 \text{ bits } = 8 \text{ bytes})
```

```
\begin{align*}
\text{sign} & \quad \text{"exponent"} & \quad \text{"significand"} \\
\downarrow & \quad \downarrow & \quad \downarrow \\
0 & \quad 1 & \quad 01111111110
\end{align*}
```

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

```
<table>
<thead>
<tr>
<th>1</th>
<th>\underline{\text{exponent}}</th>
<th>\underline{\text{significand}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\underline{2}</td>
<td>\underline{2}</td>
</tr>
</tbody>
</table>
```

A:

```
\begin{align*}
0 \times 4 & \quad 0 \times 2 \quad 1 \times 8 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \\
0 \times 4 & \quad 0 \times 2 \quad 1 \times 8 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0
\end{align*}
```

Example

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

**Significand (52 bits)**

```
\begin{align*}
\text{exponent} = 3, & \quad \text{code using 11 bits:} \\
3 + 1023 = 1026 = (10000000010)_{2}
\end{align*}
```

**double precision float (64 bits)**

```
\begin{align*}
0 \times 4 & \quad 0 \times 2 \quad 1 \times 8 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \\
0 \times 4 & \quad 0 \times 2 \quad 1 \times 8 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0 \quad 0 \times 0
\end{align*}
```

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

```
\begin{align*}
\text{sign} & \quad \text{exponent} & \quad \text{significand} \\
\downarrow & \quad \downarrow & \quad \downarrow \\
0 & \quad \downarrow & \quad \downarrow \\
\end{align*}
```

A:

```
\begin{align*}
\text{exponent code} & \quad \text{exponent value} \\
\text{unsigned exponent code} & \quad \text{exponent value} + \text{bias} \\
\hline
00000000000 & \quad \text{reserved} \\
00000000001 & \quad -1022 \\
00000000010 & \quad -1021 \\
00000000011 & \quad -1020 \\
\vdots & \quad \vdots \\
01111111111 & \quad 0 \\
10000000000 & \quad 1 \\
10000000001 & \quad 2 \\
\vdots & \quad \vdots \\
11111111110 & \quad 1023 \\
11111111111 & \quad \text{reserved}
\end{align*}
```
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
0.3 0.0000000000000004
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?

\[
\begin{align*}
2^{\frac{52}{2}} &= \left(2^{10}\right)^{5} \cdot 2^{2} \\
&= \left(10^{3}\right)^{5} \cdot 10 \\
&= 10^{14}
\end{align*}
\]

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.

Announcements

- public web page (Course outline etc)
- corequisite courses:
  COMP 206 (official)
  COMP 250 (unofficial)
  It is not recommended to do 250+206+273 together. Rather, 250+206 only, or 206+273 only.
- assignments, there will be 4 (not 3), logisim, each should take ~10 hours (still worth total of 30%)
- waiting list issues (14 x 12 + 10 = 178 seats in room)
- quiz 1: may have to sit on stairs and use a book :/
  (only 15 min)