More Integer Numbers

3.14
Integer Representation

Assume our representation has a fixed number of bits $n$ (e.g. 32 bits).

- Which 4 billion integers do we want?
  - There are an infinite number of integers less than zero and an infinite number greater than zero.

- What bit patterns should we select to represent each integer AND where the representation:
  - Does not affect the result of calculation
  - Does dramatically affect the ease of calculation

- Convert to/from human-readable representation as needed.
Integer Representation

Usual answers:

1. Represent 0 and consecutive positive integers
   - Unsigned integers
2. Represent positive and negative integers
   - Signed magnitude
   - One’s complement
   - Two’s complement
   - Biased

Unsigned and two’s complement are the most common
Signed Magnitude Integers

- A human readable way of getting both positive and negative integers.
- Not well suited to hardware implementation.
- But used with floating point.
Signed Magnitude Integers

Representation:

- Use 1 bit of integer to represent the sign of the integer
  - Sign bit is msb: 0 is “+”, 1 is “−”
- Rest of the integer is a magnitude, with same encoding as unsigned integers.
- To get the additive inverse of a number, just flip (invert, complement) the sign bit.
- Range: \(-(2^{n-1} - 1)\) to \(2^{n-1} - 1\)
Signed Magnitude - Example

If 4 bits then range is:
\[-2^3 + 1 \text{ to } 2^3 - 1\]
which is -7 to +7

Questions:
- 0101 is ?  
- -3 is ?  
- +12 is ?  
- \([-7, \ldots, -1, 0, +1, \ldots, +7]\) = 7 + 1 + 7 = 15 < 16 = 2^4
- Why?
- What problems does this cause?

\[\pm 0\]  
\[1000\]  
\[0000\]
One’s Complement

- Historically important (in other words, not used today!!!)
- Early computers built by Semour Cray (while at CDC) were based on 1’s complement integers.
- Positive integers use the same representation as unsigned.
  - 0000 is 0
  - 0111 is 7, etc
- Negation is done by taking a bitwise complement of the positive representation.
  - Complement = Invert = Not = Flip = \{0 \rightarrow 1, 1 \rightarrow 0\}
  - A logical operation done on a single bit
- Top bit is sign bit
One’s Complement Representation

To get 1’s complement of $-1$

- Take $+1$: 0001
- Complement each bit: 1110
- Don’t add or take away any bits.

Another example (4-bits):

- 1100
- This must be a negative number. To find out which, find the inverse!
- 0011 is +3
- 1100 in 1’s Complement must be?

Properties of 1’s complement:

- Any negative number will have a 1 in the MSB
- There are 2 representations for 0; 0000 and 1111

$-0$
Biased Representation

An integer representation that skews the bit patterns so as to look just like unsigned but actually represent negative numbers.

Example: 4-bit, with BIAS of $2^3$ (or called Excess 8) True value to be represented $3$
Add in the bias $+8$
Unsigned value $11$

The bit pattern of 3 in biased-8 representation will be 1011
-5 + 8 = 3

0011
Suppose we were given a biased-8 representation, 0110, to find what the number represented was:

- **Unsigned** 0110 represents 6
- Subtract out the bias -8
- True value represented -2

Operations on the biased numbers can be unsigned arithmetic but represent both positive and negative values.

How do you add two biased numbers? Subtract?
\[ x, y, b \]
\[ x + b \quad y + b \]
\[ \text{cond} \]
\[ x + b = y + b \]
\[ x + y + 2b - b \]
\[ \text{sub}_{\text{sub}} \]
\[ x + b - (y + b) \]
\[ x + b - y - b \]
\[ x - y + b \]
Biased Representation

Exercises, what are these in decimal?

25_{10} in excess 100 is: 125
52_{10, \text{excess127}} is: -75
101101_{2, \text{excess31}} is:
1101_{2, \text{excess31}} is:
Biased Representation

Where is the sign “bit” in excess notation? Bias notation used in floating-point exponents.

Choosing a bias:
To get an ~equal distribution of values above and below 0, the bias is usually $2^{n-1}$ or $2^{n-1} - 1$.

Range of bias numbers? Depends on bias, but contains $2^n$ different numbers.
Sign Extension

How to change a number with a smaller number of bits into the same number (same representation) with a larger number of bits?

This must be done frequently by arithmetic units
Sign Extension – signed magnitude

Signed magnitude:

Copy the original integer’s magnitude into the LSBs & put the original sign into the MSB, put 0’s elsewhere.

Thus for 6 bits to 8 bits

sxxxxx \rightarrow s00xxxxx
Sign Extension – 1SC

1’s complement:

1. Copy the original n-1 bits into the LSBs
2. Take the MSB of the original and copy it elsewhere

Thus for 6 bits to 8 bits:

\[ \underline{sxxxxx} \rightarrow \underline{ssssxxxx} \]
Addition: sign magnitude

- Add magnitudes only, just like unsigned addition
- Do not carry into the sign bit
- If a carry out of the MSB of magnitude then overflowed
- Add only integers of like sign ("+ to +" OR "- to -")
- Sign of the result is same as sign of the addends
Examples:

\[
\begin{array}{c}
11 \\
0 \ 0101 \ (5) \\
+ \ 0 \ 0011 \ (3)
\end{array}
\]

\[
\begin{array}{c}
+ \ 0 \ 1000 \ (8)
\end{array}
\]

\[
\begin{array}{c}
1 \ 1010 \ (-10) \\
+ \ 1 \ 0011 \ (-3)
\end{array}
\]

\[
\begin{array}{c}
\underline{+} \ 1 \ 1101 \ (-13)
\end{array}
\]

\[
\begin{array}{c}
0 \ 01011 \ (11) \\
+ \ 1 \ 01110 \ (-14)
\end{array}
\]

\[
\begin{array}{c}
\underline{+} \ 0 \ 11110 \ (1011)
\end{array}
\]

Not addition! This is subtraction
Subtraction: sign magnitude

- If signs are different, then change the problem to addition
- If the signs are the same then do subtraction
  - compare magnitudes
  - subtract smaller from larger
- if the order was switched, then switch the sign of the result

\[-3 - 4 \Rightarrow -3 + (-4)\]
Overflow in Addition

**Unsigned:** When there is a carry out of the MSB

\[
\begin{align*}
1000 & \quad (8) \\
+1001 & \quad (9) \\
\hline
10001 & \quad (1)
\end{align*}
\]
Overflow in SM Addition

Signed magnitude: When there is a carry out of the MSB of the magnitude

\[
\begin{array}{c}
1 1000 \quad (-8) \\
+1 1001 \quad (-9) \\
\hline
110001 \quad (-1)
\end{array}
\]

carry out from MSB of magnitude
Overflow in SM Subtraction

Signed magnitude: never happens when actually doing subtraction

L - S
Fractional Binary
Positional Fractions

Mesopotamians used positional fractions

\[ \sqrt{2} = \frac{1}{60} + \frac{1}{60^2} \]

1.24,51,10_{60} = 1 \times 60^0 + 24 \times 60^{-1} + 51 \times 60^{-2} +

\uparrow \uparrow \uparrow 10 \times 60^{-3}

= 1.414222

Most accurate approximation until the Renaissance
Generalized Representation

For a number “f” with “n” digits to the left and “m” to the right of the decimal place

Position is the power

Decimal point
Fractional Representation

• What is \(3E.8F_{16}\)?
  \[= 3 \times 16^1 + E \times 16^0 + 8 \times 16^{-1} + F \times 16^{-2}\]
  \[= 48 + 14 + 8/16 + 15/256\]

• How about \(10.101_2\)?
  \[= 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}\]
  \[= 2 + 0 + 1/2 + 1/8\]
Converting Decimal -> Binary fractions

- Consider left and right of the decimal point separately.
- The stuff to the left can be converted to binary as before.
- Use the following table/algorithm to convert the fraction
For $0.8_{10}$ to binary

<table>
<thead>
<tr>
<th>Fraction</th>
<th>Fraction x 2</th>
<th>Digit left of decimal point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>1.6</td>
<td>1 ← most significant ($f_{-1}$)</td>
</tr>
<tr>
<td>0.6</td>
<td>1.2</td>
<td>1</td>
</tr>
<tr>
<td>0.2</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>0.4</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>0.8</td>
<td>(it must repeat from here!!)</td>
<td></td>
</tr>
</tbody>
</table>

- Different bases have different repeating fractions.
- $0.8_{10} = 0.110011001100\ldots_2 = 0.1100_2$
- Numbers can repeat in one base and not in another.
Binary

What is $2.2_{10}$ in:

- Binary 10

0.2 × 2 = 0.4
0.4 × 2 = 0.8
0.8 × 2 = 1.6
0.6 × 2 = 1.2
0.2 × 2 = 0.4

10.0011
2

What is $2.2_{10}$ in:

- Binary

- Hex

$0.2 \times 16 = 3.2$
$0.2 \times 16 = 3.2$

$2.33$
0.625_{10} \Rightarrow 2

0.625 \times 2 = 1.25
.25 \times 2 = 0.5
.5 \times 2 = 1.0
0 \times 2 = 0

0.101
\[ 562.5_{10} \rightarrow 16 \]

\[ 0.5625 \times 16 = 9.0 \]

0.A
# Power of Two Accuracy

<table>
<thead>
<tr>
<th>Power</th>
<th>Number</th>
<th>Power</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>-17</td>
<td>7.63E-06</td>
</tr>
<tr>
<td>-1</td>
<td>0.5</td>
<td>-18</td>
<td>3.81E-06</td>
</tr>
<tr>
<td>-2</td>
<td>0.25</td>
<td>-19</td>
<td>1.91E-06</td>
</tr>
<tr>
<td>-3</td>
<td>0.125</td>
<td>-20</td>
<td>9.54E-07</td>
</tr>
<tr>
<td>-4</td>
<td>0.0625</td>
<td>-21</td>
<td>4.77E-07</td>
</tr>
<tr>
<td>-5</td>
<td>0.03125</td>
<td>-22</td>
<td>2.38E-07</td>
</tr>
<tr>
<td>-6</td>
<td>0.015625</td>
<td>-23</td>
<td>1.19E-07</td>
</tr>
<tr>
<td>-7</td>
<td>0.0078125</td>
<td>-24</td>
<td>5.96E-08</td>
</tr>
<tr>
<td>-8</td>
<td>0.00390625</td>
<td>-25</td>
<td>2.98E-08</td>
</tr>
<tr>
<td>-9</td>
<td>0.001953125</td>
<td>-26</td>
<td>1.49E-08</td>
</tr>
<tr>
<td>-10</td>
<td>0.000976563</td>
<td>-27</td>
<td>7.45E-09</td>
</tr>
<tr>
<td>-11</td>
<td>0.000488281</td>
<td>-28</td>
<td>3.73E-09</td>
</tr>
<tr>
<td>-12</td>
<td>0.000244141</td>
<td>-29</td>
<td>1.86E-09</td>
</tr>
<tr>
<td>-13</td>
<td>0.00012207</td>
<td>-30</td>
<td>9.31E-10</td>
</tr>
<tr>
<td>-14</td>
<td>6.10352E-05</td>
<td>-31</td>
<td>4.66E-10</td>
</tr>
<tr>
<td>-15</td>
<td>3.05176E-05</td>
<td>-32</td>
<td>2.33E-10</td>
</tr>
<tr>
<td>-16</td>
<td>1.52588E-05</td>
<td>-33</td>
<td>1.16E-10</td>
</tr>
</tbody>
</table>

\[10^{-5}\] \[10^{-10}\]
## Power of 16 Accuracy

<table>
<thead>
<tr>
<th>Power</th>
<th>Number</th>
<th>Power</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>-17</td>
<td>3.39E-21</td>
</tr>
<tr>
<td>-1</td>
<td>0.0625</td>
<td>-18</td>
<td>2.12E-22</td>
</tr>
<tr>
<td>-2</td>
<td>0.00390625</td>
<td>-19</td>
<td>1.32E-23</td>
</tr>
<tr>
<td>-3</td>
<td>0.000244141</td>
<td>-20</td>
<td>8.27E-25</td>
</tr>
<tr>
<td>-4</td>
<td>1.52588E-05</td>
<td>-21</td>
<td>5.17E-26</td>
</tr>
<tr>
<td>-5</td>
<td>9.53674E-07</td>
<td>-22</td>
<td>3.23E-27</td>
</tr>
<tr>
<td>-6</td>
<td>5.96086E-08</td>
<td>-23</td>
<td>2.02E-28</td>
</tr>
<tr>
<td>-7</td>
<td>3.72529E-09</td>
<td>-24</td>
<td>1.26E-29</td>
</tr>
<tr>
<td>-8</td>
<td>2.32831E-10</td>
<td>-25</td>
<td>7.89E-31</td>
</tr>
<tr>
<td>-9</td>
<td>1.45519E-11</td>
<td>-26</td>
<td>4.93E-32</td>
</tr>
<tr>
<td>-10</td>
<td>9.09495E-13</td>
<td>-27</td>
<td>3.08E-33</td>
</tr>
<tr>
<td>-11</td>
<td>5.68434E-14</td>
<td>-28</td>
<td>1.93E-34</td>
</tr>
<tr>
<td>-12</td>
<td>3.55271E-15</td>
<td>-29</td>
<td>1.2E-35</td>
</tr>
<tr>
<td>-13</td>
<td>2.22045E-16</td>
<td>-30</td>
<td>7.52E-37</td>
</tr>
<tr>
<td>-14</td>
<td>1.38778E-17</td>
<td>-31</td>
<td>4.7E-38</td>
</tr>
<tr>
<td>-15</td>
<td>8.67362E-19</td>
<td>-32</td>
<td>2.94E-39</td>
</tr>
<tr>
<td>-16</td>
<td>5.42101E-20</td>
<td>-33</td>
<td>1.84E-40</td>
</tr>
</tbody>
</table>
Binary Division Example

```
  0011111100
11 | 1011110000
   -1
   --
   011
   -1
   --
   10
   -1
   --
   00
   -0
   --
   00
   -0
   --
   00
   -0
   --
   00
```
Floating Point Numbers
Registers for real numbers usually contain 32 or 64 bits, allowing $2^{32}$ or $2^{64}$ numbers to be represented.

Which reals to represent? There are an infinite number between 2 adjacent integers. (or two reals!!)

Which bit patterns for reals selected?

Answer: use scientific notation
Consider: $A \times 10^B$, where $A$ is one digit

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$A \times 10^B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>any</td>
<td>0</td>
</tr>
<tr>
<td>1..9</td>
<td>0</td>
<td>1..9</td>
</tr>
<tr>
<td>1..9</td>
<td>1</td>
<td>10..90</td>
</tr>
<tr>
<td>1..9</td>
<td>2</td>
<td>100..900</td>
</tr>
<tr>
<td>1..9</td>
<td>-1</td>
<td>0.1..0.9</td>
</tr>
<tr>
<td>1..9</td>
<td>-2</td>
<td>0.01..0.09</td>
</tr>
</tbody>
</table>

How to do scientific notation in binary?
Standard: **IEEE 754 Floating-Point**
IEEE 754 Single Precision Floating Point Format

**SPFP**

Representation:

```
  31  30  23  22  0
  S   E   F
```

- **S** is one bit representing the sign of the number
- **E** is an 8 bit biased integer representing the exponent
- **F** is an 23-bit unsigned integer

The true value represented is: \((-1)^S \times f \times 2^e\)

- \(S = \) sign bit
- \(e = E - \text{bias}\)
- \(f = \frac{F}{2^n} + 1\)
- for single precision numbers \(n=23, \text{bias}=127\)
S, E, F are all fields within a representation. Each is just a bunch of bits.

\textbf{S} is the sign bit
\begin{itemize}
  \item \((-1)^S \rightarrow (-1)^0 = +1 \text{ and } (-1)^1 = -1\)
  \item Just a sign bit for signed magnitude
\end{itemize}

\textbf{E} is the exponent field
\begin{itemize}
  \item The E field is a biased-127 representation.
  \item True exponent is \((E - \text{bias})\)
  \item The base (radix) is always 2 (implied).
  \item Some early machines used radix 4 or 16 (IBM)
F (or M) is the fractional or mantissa field.
- It is in a strange form.
- There are 23 bits for F.
- A normalized FP number always has a leading 1.
- No need to store the one, just assume it.
- This MSB is called the HIDDEN BIT.
How to convert 64.2 into IEEE SP

1. Get a binary representation for 64.2
   - Binary of left of radix point is: \[10000000\]
   - Binary of right of radix:
     \[
     \begin{align*}
     .2 \times 2 &= 0.4 & 0 \\
     .4 \times 2 &= 0.8 & 0 \\
     .8 \times 2 &= 1.6 & 1 \\
     .6 \times 2 &= 1.2 & 1
     \end{align*}
     \]
   - Binary for .2: \[0011\]
   - 64.2 is: \[1.000000 \cdot 0011 \times 2^0\]

2. Normalize binary form
   - Produces: \[1.0000000011 \times 2^6\]
Floating Point

3. Turn true exponent into bias-127

\[
\begin{array}{c}
6 + 127 = 0 \times 00000010
\end{array}
\]

4. Put it together:

- 23-bit F is:
  \[
  \begin{array}{c}
  000000000011001100100
  \end{array}
  \]

- S E F is:

- In hex:

\[
0 \times 00000000101000000000000000000000
\]

- Since floating point numbers are always stored in normal form, how do we represent 0?
  - \(0 \times 00000000000000000000000000000000\) and \(0 \times 80000000000000000000000000000000\) represent 0.

\[
0 \times 428066666
\]
\[
6 + 127 = 133
\]
\[
\begin{array}{c}
-128 \\
\hline
5
\end{array}
\]

\[
\begin{array}{c}
1000010 \longdiv{1001} \\
\hline
1.001.2x
\end{array}
\]

\[
0.001011
\]

\[
\boxed{1.011}
\]
Other special values:

- $+ \frac{5}{0} = \infty$
- $+\infty = 0 \text{ 11111111 00000... (0x7f80 0000)}$
- $-\frac{7}{0} = -\infty$
- $-\infty = 1 \text{ 11111111 00000... (0xff80 0000)}$
- $0/0$ or $+\infty - +\infty = \text{NaN (Not a number)}$
- $\text{NaN} \neq 11111111 \text{ ???????...}$ (S is either 0 or 1, E=0xff, and F is anything but all zeroes)
- Also de-normalized numbers (beyond scope)
What is the decimal value for this SP FP number 0x4228 0000?

0 100 00100 0101000 \Rightarrow 0

128 + 4 = 132
132 - 127 = 5

1.0101000 \times 2^5
101010
42
What is $-47.625_{10}$ in SP FP format?

$101111.101 \rightarrow 1.01111110_2$

$5 + 127 = 132$

$100001000$

$1100001100 \ll 2$

$011111101100 \rightarrow 0$

$3E \ 8000$
What do floating-point numbers represent?

- Rational numbers with non-repeating expansions in the given base within the specified exponent range.
- They do not represent repeating rational or irrational numbers, or any number too small or too large.
IEEE Double Precision FP

- IEEE Double Precision is similar to SP
  - 52-bit M
  - 53 bits of precision with hidden bit
  - 11-bit E, excess 1023, representing -1023 <-> 2046
  - One sign bit
- Always use DP unless memory/file size is important unless on a microcontroller
  - SP ~ $10^{-38} \ldots 10^{38}$
  - DP ~ $10^{-308} \ldots 10^{308}$
- Be very careful of these ranges in numeric computation
Floating Point Arithmetic

Floating Point operations include
  • Addition
  • Subtraction
  • Multiplication
  • Division

They are complicated because...
Floating Point Addition

Decimal Review

\[ 9.997 \times 10^2 + 4.631 \times 10^{-1} \]

How do we do this?

1. Align decimal points
2. Add
\[ \begin{array}{c}
9.997 \times 10^2 \\
+ 0.004631 \times 10^2 \\
\hline
10.001631 \times 10^2
\end{array} \]

3. Normalize the result
   - Often already normalized
   - Otherwise move one digit
\[ 1.0001631 \times 10^3 \]

4. Possibly round result
\[ 1.000 \times 10^3 \]
Example: $0.25 + 100$ in SP FP

First step: get into SP FP if not already

\[ .25 = 0 \ 01111101 \ 00000000000000000000000000 \]
\[ 100 = 0 \ 10000101 \ 10010000000000000000000000 \]

Or with hidden bit

\[ .25 = 0 \ 01111101 \ 1 \ 00000000000000000000000000 \]
\[ 100 = 0 \ 10000101 \ 1 \ 10010000000000000000000000 \]
Second step: Align radix points

- Shifting F left by 1 bit, decreasing $e$ by 1
- Shifting F right by 1 bit, increasing $e$ by 1
- Shift F right so least significant bits fall off
- Which of the two numbers should we shift?
Second step: Align radix points cont.

Shift the .25 to increase its exponent so it matches that of 100.

\[
\begin{align*}
0.25's \ e: & \quad 01111101 - 1111111 \ (127) = -2 \\
100's \ e: & \quad 10000101 - 1111111 \ (127) = 6
\end{align*}
\]

Shift .25 by 8 then.

Easier method: Bias cancels with subtraction, so

\[
\begin{align*}
10000101 \\
- 01111101 \\
\hline
00001000
\end{align*}
\]
Carefully shifting the 0.25’s fraction

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>HB</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01111101</td>
<td>1</td>
<td>00000000000000000000000000000000 (original value)</td>
</tr>
<tr>
<td>0</td>
<td>01111110</td>
<td>0</td>
<td>10000000000000000000000000000000 (shifted by 1)</td>
</tr>
<tr>
<td>0</td>
<td>01111111</td>
<td>0</td>
<td>01000000000000000000000000000000 (shifted by 2)</td>
</tr>
<tr>
<td>0</td>
<td>10000000</td>
<td>0</td>
<td>00100000000000000000000000000000 (shifted by 3)</td>
</tr>
<tr>
<td>0</td>
<td>10000001</td>
<td>0</td>
<td>00010000000000000000000000000000 (shifted by 4)</td>
</tr>
<tr>
<td>0</td>
<td>10000010</td>
<td>0</td>
<td>00001000000000000000000000000000 (shifted by 5)</td>
</tr>
<tr>
<td>0</td>
<td>10000011</td>
<td>0</td>
<td>00000100000000000000000000000000 (shifted by 6)</td>
</tr>
<tr>
<td>0</td>
<td>10000100</td>
<td>0</td>
<td>00000010000000000000000000000000 (shifted by 7)</td>
</tr>
<tr>
<td>0</td>
<td>10000101</td>
<td>0</td>
<td>00000001000000000000000000000000 (shifted by 8)</td>
</tr>
</tbody>
</table>
Third Step: Add fractions with hidden bit

\[
\begin{align*}
0 & \quad 10000101 \quad 1 \quad 10010000000000000000000000000000 \\
+ & \quad 0 \quad 10000101 \quad 0 \quad 0000000100000000000000000 \\
\hline
0 & \quad 10000101 \quad 1 \quad 10010001000000000000000000000000
\end{align*}
\] (100) (.25)

Fourth Step: Normalize the result

- Get a ‘1’ back in hidden bit
- Already normalized most of the time
- Remove hidden bit and finished
Normalization example

\[
\begin{array}{ccccc}
S & E & HB & F \\
0 & 011 & 1 & 1100 \\
+ & 0 & 011 & 1 & 1011 \\
\hline
0 & 011 & 11 & 0111 \\
\end{array}
\]

Need to shift so that only a 1 in HB spot

\[
0 & 100 & 1 & 1011 & 1 \rightarrow \text{discarded}
\]
Floating Point Subtraction

- Mantissa's are sign-magnitude
- Watch out when the numbers are close

\[
\begin{align*}
1.23455 \times 10^2 \\
- 1.23456 \times 10^2
\end{align*}
\]

- A many-digit normalization is possible
  This is why FP addition is in many ways more difficult than FP multiplication
Steps to do subtraction

1. Align radix points
2. Perform sign-magnitude operand swap if needed
   • Compare magnitudes (with hidden bit)
   • Change sign bit if order of operands is changed.
3. Subtract
4. Normalize
5. Round
Simple Example:

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>HB</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>011</td>
<td>1</td>
<td>1011</td>
<td>smaller</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>011</td>
<td>1</td>
<td>1101</td>
</tr>
</tbody>
</table>

switch order and make result negative

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>HB</th>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>011</td>
<td>1</td>
<td>1101</td>
<td>bigger</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>011</td>
<td>1</td>
<td>1011</td>
</tr>
<tr>
<td>1</td>
<td>011</td>
<td>0</td>
<td>0010</td>
<td></td>
</tr>
</tbody>
</table>

1 0000 1 0000  switched sign, renormalized
Floating Point Multiplication

Decimal example:

$$3.0 \times 10^1 \times 5.0 \times 10^2$$

1. Multiply mantissas
   $$3.0 \times 5.0 = 15.00$$

2. Add exponents
   $$1 + 2 = 3$$

3. Combine
   $$15.00 \times 10^3$$

4. Normalize if needed
   $$1.50 \times 10^4$$

How do we do this?
Multiplication in binary (4-bit F)

\[
\begin{array}{c}
0 \ 10000100 \ 0100 \\
\times \ 1 \ 00111100 \ 1100 \\
\hline
\end{array}
\]

Step 1: Multiply mantissas (put hidden bit back first!!)

\[
\begin{array}{c}
1.0100 \\
\times \ 1.1100 \\
\hline
00000 \\
00000 \\
10100 \\
10100 \\
+ \ 10100 \\
\hline
10001100000 \\
\end{array}
\]
Second step: Add exponents, subtract extra bias.

\[
\begin{align*}
10000100 & \quad + \quad 00111100 \\
\underline{11000000} & \quad - \quad 01111111 \quad (127)
\end{align*}
\]

Third step: Renormalize, correcting exponent

\[
\begin{align*}
1 & \quad 01000001 \quad 10.00110000 \\
\text{Becomes} & \\
1 & \quad 01000010 \quad 1.000110000
\end{align*}
\]

Fourth step: Drop the hidden bit

\[
\begin{align*}
1 & \quad 01000010 \quad 000110000 \\
= & \quad 0xA10C0000
\end{align*}
\]
Multiply these SP FP numbers together

\[ 0x49FC0000 \times 0x4BE00000 \]
Floating Point Division

• True division
  • Unsigned, full-precision division on mantissas
    • This is much more costly (e.g. 4x) than mult.
  • Subtract exponents
• Faster division
  • Newton’s method to find reciprocal
  • Multiply dividend by reciprocal of divisor
  • May not yield exact result without some work
  • Similar speed as multiplication
• Not covered in this class!
Floating Point Summary

- Has 3 portions, S, E, F/M
- Do conversion in parts
- Arithmetic is signed magnitude
- Subtraction could require many shifts for renormalization
- Multiplication is easier since do not have to match exponents