Lecture 10: Division, Floating Point

• Today’s topics:
  ▪ Division
  ▪ IEEE 754 representations
Division

<table>
<thead>
<tr>
<th>Divisor</th>
<th>1000_{ten}</th>
<th>1001_{ten}</th>
<th>Quotient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dividend</td>
<td>1001010_{ten}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

At every step,

- shift divisor right and compare it with current dividend
- if divisor is larger, shift 0 as the next bit of the quotient
- if divisor is smaller, subtract to get new dividend and shift 1 as the next bit of the quotient
## Division

<table>
<thead>
<tr>
<th>Divisor</th>
<th>1000\text{_{ten}}</th>
<th>\underline{1001}\text{_{ten}}</th>
<th>Quotient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\underline{1001010}\text{_{ten}}</td>
<td>Dividend</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
0001001010 & \rightarrow 0001001010 \\
100000000000 & \rightarrow 0001000000 \\
& \rightarrow 0000100000 \\
& \rightarrow 0000001000 \\
\text{Quo: } 0 & \rightarrow 000001 \\
& \rightarrow 0000010 \\
& \rightarrow 000001001
\end{align*}
\]

At every step,
- shift divisor right and compare it with current dividend
- if divisor is larger, shift 0 as the next bit of the quotient
- if divisor is smaller, subtract to get new dividend and shift 1 as the next bit of the quotient
Divide Example

- Divide $7_{\text{ten}} (0000\ 0111_{\text{two}})$ by $2_{\text{ten}} (0010_{\text{two}})$

<table>
<thead>
<tr>
<th>Iter</th>
<th>Step</th>
<th>Quot</th>
<th>Divisor</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial values</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Divide Example

- Divide $7_{10} (0000\ 0111_{two})$ by $2_{10} (0010_{two})$

<table>
<thead>
<tr>
<th>Iter</th>
<th>Step</th>
<th>Quot</th>
<th>Divisor</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial values</td>
<td>0000</td>
<td>0010 0000</td>
<td>0000 0111</td>
</tr>
<tr>
<td>1</td>
<td>Rem = Rem – Div</td>
<td>0000</td>
<td>0010 0000</td>
<td>1110 0111</td>
</tr>
<tr>
<td></td>
<td>Rem &lt; 0 ➞ +Div, shift 0 into Q</td>
<td>0000</td>
<td>0010 0000</td>
<td>0000 0111</td>
</tr>
<tr>
<td></td>
<td>Shift Div right</td>
<td>0000</td>
<td>0001 0000</td>
<td>0000 0111</td>
</tr>
<tr>
<td>2</td>
<td>Same steps as 1</td>
<td>0000</td>
<td>0001 0000</td>
<td>1111 0111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000</td>
<td>0001 0000</td>
<td>0000 0111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000</td>
<td>0000 1000</td>
<td>0000 0111</td>
</tr>
<tr>
<td>3</td>
<td>Same steps as 1</td>
<td>0000</td>
<td>0000 0100</td>
<td>0000 0111</td>
</tr>
<tr>
<td>4</td>
<td>Rem = Rem – Div</td>
<td>0000</td>
<td>0000 0100</td>
<td>0000 0011</td>
</tr>
<tr>
<td></td>
<td>Rem &gt;= 0 ➞ shift 1 into Q</td>
<td>0001</td>
<td>0000 0100</td>
<td>0000 0011</td>
</tr>
<tr>
<td></td>
<td>Shift Div right</td>
<td>0001</td>
<td>0000 0010</td>
<td>0000 0011</td>
</tr>
<tr>
<td>5</td>
<td>Same steps as 4</td>
<td>0011</td>
<td>0000 0001</td>
<td>0000 0001</td>
</tr>
</tbody>
</table>
Hardware for Division

A comparison requires a subtract; the sign of the result is examined; if the result is negative, the divisor must be added back.

Similar to multiply, results are placed in Hi (remainder) and Lo (quotient).
Efficient Division

Source: H&P textbook
Divisions involving Negatives

• Simplest solution: convert to positive and adjust sign later

• Note that multiple solutions exist for the equation:
  Dividend = Quotient x Divisor + Remainder

  +7  div  +2  Quo =  Rem =
  -7  div  +2  Quo =  Rem =
  +7  div  -2  Quo =  Rem =
  -7  div  -2  Quo =  Rem =
Divisions involving Negatives

• Simplest solution: convert to positive and adjust sign later

• Note that multiple solutions exist for the equation:
  Dividend = Quotient x Divisor + Remainder

  +7 div +2  Quo = +3  Rem = +1
  -7 div +2  Quo = -3  Rem = -1
  +7 div -2  Quo = -3  Rem = +1
  -7 div -2  Quo = +3  Rem = -1

  Convention: Dividend and remainder have the same sign
  Quotient is negative if signs disagree
  These rules fulfil the equation above
Floating Point

• Normalized scientific notation: single non-zero digit to the left of the decimal (binary) point – example: \(3.5 \times 10^9\)

\[1.010001 \times 2^{-5}_{\text{two}} = (1 + 0 \times 2^{-1} + 1 \times 2^{-2} + \ldots + 1 \times 2^{-6}) \times 2^{-5}_{\text{ten}}\]

• A standard notation enables easy exchange of data between machines and simplifies hardware algorithms – the IEEE 754 standard defines how floating point numbers are represented
# Sign and Magnitude Representation

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

- More exponent bits → wider range of numbers (not necessarily more numbers – recall there are infinite real numbers)
- More fraction bits → higher precision
- Register value = \((-1)^S \times F \times 2^E\)
- Since we are only representing normalized numbers, we are guaranteed that the number is of the form 1.xxxx..
  Hence, in IEEE 754 standard, the 1 is implicit
  Register value = \((-1)^S \times (1 + F) \times 2^E\)
## Sign and Magnitude Representation

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

- **Sign**: indicates the sign of the number (0 for positive, 1 for negative).
- **Exponent**: used to scale the significand.
- **Fraction**: the fractional part of the number.

- Largest number that can be represented: \(2.0 \times 2^{128} = 2.0 \times 10^{38}\) (not really – see upcoming details)
- Smallest number that can be represented: \(1.0 \times 2^{-127} = 2.0 \times 10^{-38}\) (not really – see upcoming details)
- Overflow: when representing a number larger than the max;
  Underflow: when representing a number smaller than the min

### Double precision format: occupies two 32-bit registers:

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>11 bits</td>
<td>52 bits</td>
</tr>
</tbody>
</table>

Largest: \(2^{1023}\), Smallest: \(2^{-1022}\)
Details

• The number “0” has a special code so that the implicit 1 does not get added: the code is all 0s (it may seem that this takes up the representation for 1.0, but given how the exponent is represented, that’s not the case) (see discussion of denorms in the textbook)

• The largest exponent value (with zero fraction) represents +/- infinity

• The largest exponent value (with non-zero fraction) represents NaN (not a number) – for the result of 0/0 or (infinity minus infinity)

• Note that these choices impact the smallest and largest numbers that can be represented
Exponent Representation

• To simplify sort, sign was placed as the first bit

• For a similar reason, the representation of the exponent is also modified: in order to use integer compares, it would be preferable to have the smallest exponent as 00...0 and the largest exponent as 11...1

• This is the biased notation, where a bias is subtracted from the exponent field to yield the true exponent

• IEEE 754 single-precision uses a bias of 127 (since the exponent must have values between -127 and 128)...double precision uses a bias of 1023

Final representation: \((-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} – \text{Bias})}\)
Value inf
Value NAN
Highest value $\sim 2 \times 2^{127}$

Value 1

Smallest Norm $\sim 2 \times 2^{-126}$
Largest Denorm $\sim 1 \times 2^{-126}$
Smallest Denorm $\sim 2^{-149}$
Value 0

2 special cases up top that use the reserved exponent field of 255
Exponent field < 127, i.e., after subtracting bias, they are negative exponents, representing numbers $< 1$

Special case with exponent field 0, used to represent denorms, that help us gradually approach 0

Same rules as above, but the sign bit is 1
Same magnitudes as above, but negative numbers
Examples

Final representation: \((-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}\)

• Represent \(-0.75_{\text{ten}}\) in single and double-precision formats

  Single: \((1 + 8 + 23)\)

  Double: \((1 + 11 + 52)\)

  Remember:
  
  ![Diagram showing the relationship between True exponent and Exponent in register]

• What decimal number is represented by the following single-precision number?
  \(1 1000 0001 01000...0000\)
Final representation: \((-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}\)

• Represent \(-0.75_{\text{ten}}\) in single and double-precision formats

Single: \((1 + 8 + 23)\)
1 0111 1110 1000...000

Double: \((1 + 11 + 52)\)
1 0111 1111 110 1000...000

• What decimal number is represented by the following single-precision number?
1 1000 0001 01000...0000
-5.0
Example 2

Final representation: \((-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}\)

- Represent \(36.90625_{\text{ten}}\) in single-precision format

\[
\begin{align*}
36 / 2 &= 18 \text{ rem } 0 \quad &0.90625 \times 2 &= 1.81250 \\
18 / 2 &= 9 \text{ rem } 0 \quad &0.8125 \times 2 &= 1.6250 \\
9 / 2 &= 4 \text{ rem } 1 \quad &0.625 \times 2 &= 1.250 \\
4 / 2 &= 2 \text{ rem } 0 \quad &0.25 \times 2 &= 0.50 \\
2 / 2 &= 1 \text{ rem } 0 \quad &0.5 \times 2 &= 1.00 \\
1 / 2 &= 0 \text{ rem } 1 \quad &0.0 \times 2 &= 0.0
\end{align*}
\]

36 is 100100

0.90625 is 0.1110100...0
Example 2

Final representation: $(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

We’ve calculated that $36.90625_{\text{ten}} = 100100.1110100...0$ in binary
Normalized form = $1.001001110100...0 \times 2^5$
  (had to shift 5 places to get only one bit left of the point)

The sign bit is 0 (positive number)
The fraction field is 001001110100...0 (the 23 bits after the point)
The exponent field is 5 + 127 (have to add the bias) = 132,
  which in binary is 10000100

The IEEE 754 format is 0 10000100 001001110100.....0
  sign  exponent  23 fraction bits