Divide: Paper & Pencil

<table>
<thead>
<tr>
<th>Divisor 1000</th>
<th>1101010 Dividend</th>
</tr>
</thead>
</table>

Dividend = Quotient \times Divisor + Remainder

- See how big a number can be subtracted, creating quotient bit on each step
  - Binary \(\Rightarrow 1 \times \text{divisor} \text{ or } 0 \times \text{divisor}\)

CSE 141  Dean Tullsen

Divide Algorithm

Version 1

- Takes \(n+1\) steps for \(n\)-bit Quotient & Rem.

<table>
<thead>
<tr>
<th>Quotient</th>
<th>Divisor</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0011</td>
<td>0000</td>
</tr>
<tr>
<td>0111</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

1. Subtract the Divisor register from the Remainder register, and place the result in the Remainder register.

2a. Shift the Quotient register to the left, setting the new rightmost bit to 1.

2b. Restore the original value by adding the Divisor register to the Remainder register, and place the sum in the Remainder register. Also shift the Quotient register to the left, setting the new least significant bit to 0.

3. Shift the Divisor register right 1 bit.

CSE 141  Dean Tullsen

Divide HARDWARE

Version 1

- 64-bit Divisor reg, 64-bit ALU, 64-bit Remainder reg, 32-bit Quotient reg

Version 3

- 32-bit Divisor reg, 32-bit ALU, 64-bit Remainder reg, (0-bit Quotient reg)
Observations on Divide
Version 3

• Same Hardware as: just need ALU to add or subtract, and 63-bit register to shift left or shift right
• and registers in MIPS combine to act as 64-bit register for multiply and divide
• Signed Divides: Simplest is to remember signs, make positive, and complement quotient and remainder if necessary
  – Note:
  – Note:

So Far

• Can do logical, add, subtract, multiply, divide, ...
• But.......
  – what about fractions?
  – what about really large numbers?

Binary Fractions

\[ 1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \]
so...
\[ 101.011_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \]
e.g.,
\[ .75 = 3/4 = 3/2^2 = 1/2 + 1/4 = .11 \]

Recall Scientific Notation

\[ +6.02 \times 10^{23} \]
\[ 1.673 \times 10^{-24} \]

Issues:
- Arithmetic (+, -, *, /)
- Representation, Normal form
- Range and Precision
- Rounding
- Exceptions (e.g., divide by zero, overflow, underflow)
- Errors
- Properties (negation, inversion, if A = B then A - B = 0)
Floating-Point Numbers

Representation of floating point numbers in IEEE 754 standard:

<table>
<thead>
<tr>
<th>single precision</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>sign</td>
<td>E</td>
</tr>
<tr>
<td>mantissa:</td>
<td>M</td>
</tr>
<tr>
<td>(actual exponent</td>
<td>E</td>
</tr>
<tr>
<td>is e = E - 127)</td>
<td></td>
</tr>
</tbody>
</table>

\[ N = (-1)^S \times 2^{E-127} \times (1.M) \]

0 < E ≤ 223

- 0.00000000 0 . . . 0
- 0.10111111 10 . . . 0
- 0.10000011 01000101 X 2^8
- 0.00110110100 . . . 0
= 0.10111111 100110100 . . . X 2^-3

- range of about 2 X 10^-38 to 2 X 10^{38}
- always normalized (so always leading 1, thus never shown)
- special representation of 0 (E = 00000000) (why?)
- can do integer compare for greater-than, sign

CSE 141
Dean Tullsen

Double Precision Floating Point

Representation of floating point numbers in IEEE 754 standard:

<table>
<thead>
<tr>
<th>double precision</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>sign</td>
<td>E</td>
</tr>
<tr>
<td>mantissa:</td>
<td>M</td>
</tr>
<tr>
<td>(actual exponent</td>
<td>E</td>
</tr>
<tr>
<td>is e = E - 1023)</td>
<td></td>
</tr>
</tbody>
</table>

\[ N = (-1)^S \times 2^{E-1023} \times (1.M) \]

0 < E ≤ 2^{1048}

- 52 (+1) bit mantissa
- range of about 2 X 10^-308 to 2 X 10^{308}

CSE 141
Dean Tullsen

Floating Point Addition

- How do you add in scientific notation?

9.962 x 10^4 + 5.231 x 10^3

- Basic Algorithm
  1. 
  2. 
  3. 
  4. 

CSE 141
Dean Tullsen

FP Addition Hardware

CSE 141
Dean Tullsen
Floating Point Multiplication

- How do you multiply in scientific notation?
  \((9.9 \times 10^4)(5.2 \times 10^2) = 5.148 \times 10^7\)

- Basic Algorithm
  1.
  2.
  3.
  4.
  5.

FP Accuracy

- Extremely important in scientific calculations
- Very tiny errors can accumulate over time
- IEEE 754 FP standard has four rounding modes
  - always round up (toward +\(\infty\))
  - always round down (toward -\(\infty\))
  - truncate
  - round to nearest
    => in case of tie, round to nearest even
- Requires in intermediate representations

Extra Bits for FP Accuracy

- *Guard bits* -- bits to the right of the least significant bit of the significand computed for use in normalization (could become significant at that point) and rounding.
- IEEE 754 has three extra bits and calls them , and .

Key Points

- Multiplication and division take much longer than addition, requiring multiple addition steps.
- Floating Point extends the range of numbers that can be represented, at the expense of precision (accuracy).
- FP operations are very similar to integer, but with pre- and post-processing.
- Rounding implementation is critical to accuracy over time.