Integer and FP Arithmetic

CPS 104
Lecture 11

Outline
• Integer multiply & divide
• Floating Point Arithmetic
• Review Storage elements
• Building a Data Path

Reading
• Chapter 4.8, 5.1-5.3

Arithmetic
• Integer Addition---Done
• Integer Multiplication (Ch 4.6)
• Integer Division (Ch 4.7)
• Floating Point Addition (Ch 4.8)
• Floating Point Multiplication (Ch 4.8)

Integer Multiplication
• Product = Multiplicand \times Multiplier
• Example: 0011_{\text{ten}} \times 0101_{\text{ten}}

Multiplication Algorithm #1
• From Right-Left:
  ➢ If multiplier digit = 1: add (shifted) copy of multiplicand to result.
  ➢ If multiplier digit = 0: add 0 to result.
• 32 steps when multiplier is 32-bit number.
• Example: \(3_{10} \times 5_{10}\) or 0011\(_{2}\) \times 0101\(_{2}\)
Product = 00001111\(_{2}\)
Multiplication Hardware #1
- Multiplicand starts in right half of register
- MIPS: 64-bit product in Hi & Lo Regs
  - Move from Lo (mflo) to get 32-bit product
  - Move from hi (mfhi) to get upper 32-bits & test for overflow

Multiplication Hardware #2
- Shift Multiplicand Left ~ Shift Product Right
- Only need 32 bits for multiplicand

Multiplication Algorithm #2
1. Test
2. Shift the Product register right 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
- No: < 32 repetitions
- Yes: 32 repetitions

Booth Encoding
- Observation:
  - Can write number as difference of two numbers.
  - In particular: Can replace a string of 1s with initial subtract when we see a 1, and then an add when we see the bit AFTER the last 1
- Example 1: 7_{10}
  - 7_{10} = -110 + 8_{10}
  - 0111_{2} = -0001_{2} + 1000_{2}
- Example 2: 110_{10} = 01101110_{2}
  - 110_{10} = (-2_{10} + 16_{10}) + (-32_{10} + 128_{10})
  - 01101110_{2} = (-00000010_{2} + 00010000_{2}) + (-00100000_{2} + 10000000_{2})
- Works for signed numbers as well!

Booth's Algorithm
- Similar to previous multiply algorithm.
- (Current, Previous) bits of Multiplier:
  - 0.0: middle of string of 0s; do nothing
  - 0.1: end of a string of 1s; add multiplicand
  - 1.0: start of string of 1s; subtract multiplicand
  - 1.1: middle of string of 1s; do nothing
- Shift Product/Multiplier right 1 bit (as before)

Signed Multiplication
- Convert negative numbers to positive and remember the original signs.
- In 2s-complement, can multiply directly using Booth's Algorithm.
  - Sign extend when shifting.
**Integer Division**

- Dividend = Quotient x Divisor + Remainder
- Example: $1,001,010_{\text{ten}} / 1000_{\text{ten}}$

```
1 0 0 1 0
Divisor 1000
1 0 0 1 0 Quotient
1 0 1 0 Division
1 0 1 0
1 0 0 0 0
1 0 0 1 Remainder
```

**Division Hardware #1**

- Divisor starts in left half of divisor register
- Remainder initialized to dividend

1. Subtract divisor from dividend
2. If result positive:
   - Shift in 1 to Quotient right bit
   - Restore value by adding divisor to Remainder
3. Shift divisor right 1 bit
4. If 33rd iteration stop else goto 1

**Division (contd.)**

- Similar to multiplication
  - Shift remainder left instead of shifting divisor right
  - Combine quotient register with right half of remainder register
  - MIPS: Hi contains remainder, Lo contains quotient

- Signed Division
  - Remember the signs and negate quotient if different.
  - Make sign of remainder match the dividend

- Same hardware can be used for both multiply and divide.
  - Need 64-bit register that can shift left and right
  - ALU that adds or subtracts
  - Optimizations possible

**Review: FP Representation**

Numbers are represented by:

$$X = (-1)^S \times 2^{E-127} \times 1.M$$

- $S$ := 1-bit field ; Sign bit
- $E$ := 8-bit field; Exponent: Biased integer, $0 \leq E \leq 255$.
- $M$ := 23-bit field; Mantissa: Normalized fraction with
  hidden 1 (don’t actually store it)

Single precision floating point number uses 32-bits for representation

```
<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-bit</td>
<td>23-bit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- $S = 0$: Normalized: $1.0015 	imes 10^2$
- $S = 1$: Round: $1.002 	imes 10^2$

**Floating Point Representation**

- The mantissa represents a fraction using binary notation:
  $$M = \cdot s_1 s_2 s_3 \ldots = 1.0 + s_1 \times 2^{-1} + s_2 \times 2^{-2} + s_3 \times 2^{-3} + \ldots$$

- Example: $X = -0.75_{\text{ten}}$ in single precision ($-1/2 + 1/4$)

```
S = 1 ; Exp = 126_{\text{ten}} = 0111 1110_{\text{bin}} ;
M = 100 0000 0000 0000 0000 0000_{\text{bin}}
```

```
<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111 1110</td>
<td>00 0000 0000 0000 0000 0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>$E$</td>
<td>$M$</td>
<td></td>
</tr>
</tbody>
</table>
```

**FP Addition**

- Example: $5.610 \times 10^1 + 9.999 \times 10^1$
- Step 1:
  - Align decimal point: $0.016 \times 10^2 + 9.999 \times 10^1$
- Step 2:
  - Add: $10.015 \times 10^1$
- Step 3:
  - Normalize: $1.0015 \times 10^2$
- Step 4:
  - Round: $1.002 \times 10^2$

May need to repeat steps 3 and 4 if result not normal after rounding. (Renormalization)
Floating Point Addition

1. Compare the exponents of the two numbers. Shift the smaller number to the right until its exponent would match the larger exponent.
2. Add the significands.
3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent.
4. Round the significand to the appropriate number of bits.
5. Still normalized?
   - Yes
   - No
   - Overflow or underflow?

Arithmetic Unit for FP Addition

1. Compare the exponents of the two numbers. Shift the smaller number to the right until its exponent would match the larger exponent.
2. Add the significands.
3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent.
4. Round the significand to the appropriate number of bits.
5. Still normalized?
   - Yes
   - No

FP Multiplication

1. Add biased exponents, subtract bias.
2. Multiply significands.
3. Normalize product.
4. Round significand.
5. Compute sign of product.

- \(.5 \times -0.75 \Rightarrow 1.000x2^{-1} \times 1.100x2^{-2}\)

Example FP Multiply

- \(.5 \times -0.75 \Rightarrow 1.000x2^{-1} \times 1.100x2^{-2}\)
- With Bias \(1.000x2^{126} \times 1.100x2^{124}\)
  1. \(126+124-127 = 123\)
  2. 
  3. Normalize product: \(1.1000000x2^{123}\)
  4. Round: \(1.100x2^{123}\)
  5. Compute Sign: different so result is neg \(-1.100x2^{123} = -.375\)

Accuracy

- Is \((x+y)+z = x + (y+z)\)?
- Computer numbers have limited size => limited precision.
- Rounding Errors
  - Example: \(2.56 \times 10^9 + 2.34 \times 10^9\), using 3 significant digits
  - Align decimal points (exponents, shift smaller)
    - \(2.34\)
    - \(0.0236\)
    - \(2.36\)
Rounding

- Rounding with Guard & Round bits
- Example: $2.56 \times 10^6 + 2.34 \times 10^2$, using 3 significant digits
- Align decimal points (exponents, shift smaller)
  
  $$
  \begin{array}{c}
  2.34 \\
  0.0256 \\
  2.3656
  \end{array}
  $$

  - Guard 5, Round 6
- Round: $2.37 \times 10^0$
- Without guard & round bits, result: $2.36 \times 10^0$
- Error of 1 Unit in the least significant position
- Why 2 bits?
  
  > Product could have leading 0, so shift left when normalizing

Arithmetic Exceptions

- Conditions
  
  > Overflow
  > Underflow
  > Division by zero
- Special floating point values
  
  > +infinity (e.g. 1/0)
  > -infinity (e.g. -1/0)
  > Nan (Not A Number) (e.g. 0/0, infinity/infinity, sq. root of -1)

Summary

- Integer Multiplication
- Integer Division
- Floating Point Addition
- Floating Point Multiplication
- Accuracy (Guard and Round Bits)

Next Time

- Review storage elements
- Datapath, Chapter 5...