Integer and FP Arithmetic

CPS 104
Lecture 11

Administrivia

• Homework #4, Due next Thursday
• MIPS Simulator Due April 14
  ➢ Groups: email yike@cs, groups of 2 or 3

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 x Multiplier

Example: $0011_{\text{ten}} \times 0101_{\text{ten}}$

<table>
<thead>
<tr>
<th>Multiplicand</th>
<th>0 0 1 1_{\text{ten}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier</td>
<td>0 1 0 1_{\text{ten}}</td>
</tr>
<tr>
<td>Product</td>
<td>0 0 0 0 0 0 1 1_{\text{ten}}</td>
</tr>
</tbody>
</table>
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 Algorithm #1**

1. **Start**
   - **Multiplier0 = 0**
   - **Multiplier0 = 1**
2. Test Multiplier0
3. 1. Add multiplicand to product and place the result in Product register
   - **1a. Shift the Multiplicand register left 1 bit**
   - **2. Shift the Multiplier register right 1 bit**
4. **32nd repetition?**
   - No: < 32 repetitions
   - Yes: 32 repetitions
5. **Done**
### 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 #1 Diagram](image)

---

### Multiplication Hardware #2

- **Shift Multiplicand** Left ~ **Shift Product** Right
- **Only need 32 bits** for multiplicand

![Multiplication Hardware #2 Diagram](image)

---

- **Possible to combine multiplier and product registers**
Multiplication Algorithm #2

1. Test Multiplier0
   1a. Add multiplicand to the left half of the product and place the result in the left half of the Product register
2. Shift the Product register right 1 bit
3. Shift the Multiplier register right 1 bit

32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions

Done

Figure Copyright Morgan Kaufmann

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: $7_{10}$
  - $7_{10} = -1_{10} + 8_{10}$
  - $0111_2 = -0001_2 + 1000_2$

- 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: $1,001,010_{ten} / 1000_{ten}$

\[
\begin{array}{c|cccc}
\text{Dividend} & 1 & 0 & 0 & 1_{ten} \\
\hline
\text{Divisor} & 1 & 0 & 0 & 1_{ten} \\
\text{Quotient} & 1 & 0 & 0 & 1_{ten} \\
\text{Remainder} & -1 & 0 & 0 & 1_{ten}
\end{array}
\]

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
else
   - restore value by adding divisor to Remainder
   - Shift in 0 to Quotient right bit
3. Shift divisor right 1 bit
4. If 33rd iteration stop else goto 1

![Division Hardware Diagram](Figure Copyright Morgan Kaufmann)
**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>22</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>exp</td>
<td>23-bit</td>
<td>Mantissa</td>
</tr>
</tbody>
</table>

- **8-bit**
Floating Point Representation

• The mantissa represents a fraction using binary notation:
  \[ M = . s_1, s_2, s_3 \ldots = 1.0 + s_1 \times 2^{-1} + s_2 \times 2^{-2} + s_3 \times 2^{-3} + \ldots \]

• Example: \( X = -0.75_{10} \) in single precision \((-\frac{1}{2} + \frac{1}{4})\)

\[-0.75_{10} = -0.112 = (-1) \times 1.12 \times 2^{-1} = (-1) \times 1.12 \times 2^{126-127} \]
\[ S = 1 \; \text{ ; Exp} = 126_{10} = 01111110_2 \; ; \]
\[ M = 10000000000000000000_2 \]

\[ X = \begin{array}{c|c|c|c|c|c|c|c|c|c|c} 31 & 30 & 23 & 22 & 0 \\
\hline
s & E & M
\end{array} \]

FP Addition

• Example:
  \[ 1.610 \times 10^{-1} + 9.999 \times 10^1 \]

• Step 1:
  \[ \text{Align decimal point: } 0.016 \times 10^1 + 9.999 \times 10^1 \]

• Step 2:
  \[ \text{Add: } 10.015 \times 10^1 \]

• Step 3:
  \[ \text{Normalize: } 1.0015 \times 10^2 \]

• Step 4:
  \[ \text{Round: } 1.002 \times 10^2 \]

• 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 increasing the exponent or shifting left and decreasing the exponent.

4. Round the significand to the appropriate number of bits
   - Overflow or underflow?
     - Yes → Exception
     - No → Still normalized?
       - Yes → Done
       - No → Still normalized?

Arithmetic Unit for FP Addition

- Sign, Exponent, Significand
- Sign, Exponent, Significand
- Compare exponents
- Shift smaller number right
- Add
- Normalize
- Round
FP Multiplication

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

• .5 x -.75 => 1.000x2⁻¹ x 1.100x2⁻²

Example FP Multiply

• .5 x -.75 => 1.000x2⁻¹ x 1.100x2⁻²
• With Bias 1.000x2¹²⁶ x 1.100x2¹²⁴
1. 126+124-127 = 123
2. 
   \[
   \begin{array}{c}
   & 1.000 \\
   + & 1.100 \\
   \hline
   & 0000 \\
   \hline
   & 0000 \\
   \hline
   & 1000 \\
   \hline
   & 1000 \\
   \hline
   & 1.100000
   \end{array}
   \]
3. Normalize product: 1.1000000x2¹²³
4. Round: 1.100x2¹²³
5. Compute Sign: different so result is neg
   - 1.100x2¹²³ = -.375
FP Multiplication

1. Add the biased exponents of the two numbers, subtracting the bias from the sum to get the new biased exponent
2. Multiply the significands
3. Normalize the product if necessary, shifting it right and incrementing the exponent
4. Round the significand to the appropriate number of bits
5. Set the sign of the product to positive if the signs of the original operands are the same; if they differ, make the sign negative

Accuracy

- Is \((x+y)+z = x + (y+z)\)?
- Computer numbers have limited size => limited precision.
- Rounding Errors

- Example:
  \[2.56 \times 10^0 + 2.34 \times 10^2, \text{ using 3 significant digits}\]
  - Align decimal points (exponents, shift smaller)
    - 2.34
    - 0.0256
    - 2.36
Rounding

- Rounding with Guard & Round bits
- Example: \(2.56 \times 10^0 + 2.34 \times 10^2\), 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…