Integer and FP Arithmetic

Computer Science 104

Administrivia

- Homework #3, Due Friday
- Projects Due Dec 7
  - Groups on course web page

Outline

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

Reading

- Today: Chapter 3.6 – 3.9
- Next: Data path Chapter 5.1 – 5.3, Control 5.4

Booth's 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)
- Ignore overflow

Signed Multiplication

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

Compute using Booth's algorithm

- 11000111 x 01101110
- (8-bit 2's complement numbers)
- Remember leading 1's for negative numbers, leading 0's for positive numbers
- Example: 4- bit -3*-2

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

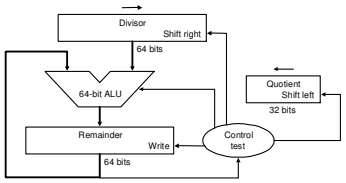


Figure Copyright Morgan Kaufmann

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:

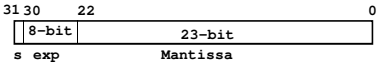
$$X = (-1)^s \times 2^{E-127} \times 1.M$$

S := 1-bit field ; Sign bit

E := 8-bit field; Exponent: Biased integer, $0 \le E \le 255$.

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



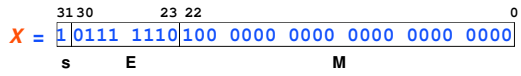
Floating Point Representation

- The mantissa represents a fraction using binary notation:
  $M = . s_1, s_2, s_3 \ldots = 1.0 + s_1*2^{-1} + s_2*2^{-2} + s_3*2^{-3} + \ldots$
- Example: $X = -0.75_{10}$ in single precision ($-(1/2 + 1/4)$)

$-0.75_{10} = -0.11_2 = (-1) \times 1.1_2 \times 2^{-1} = (-1) \times 1.1_2 \times 2^{126-127}$

$S = 1$; $Exp = 126_{10} = 0111\ 1110_2$;

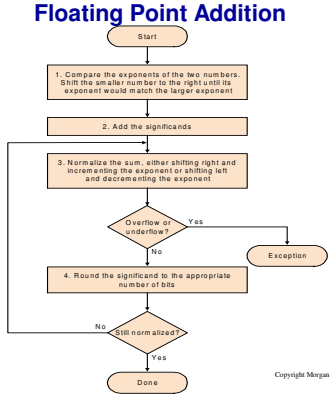
$M = 100\ 0000\ 0000\ 0000\ 0000\ 0000_2$



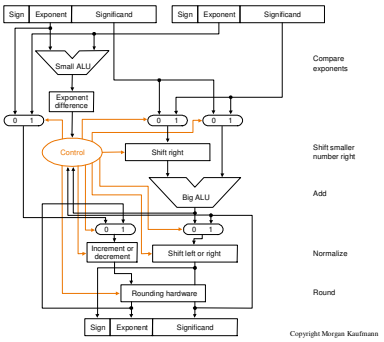
FP Addition

- Example: Assume only 4 digits
  - $1.610 \times 10^{-1} + 9.999 \times 10^{1}$
- Step 1:
  - Align decimal point: $0.016 \times 10^{1} + 9.999 \times 10^{1}$
- Step 2:
  - Add: 10.015×10^{1}
- Step 3:
  - Normalize: 1.0015×10^{2}
- Step 4:
  - Round: 1.002×10^{2}
- May need to repeat steps 3 and 4 if result not normal after rounding. (renormalization)

Floating Point Addition



Arithmetic Unit for FP Addition



FP Multiplication

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

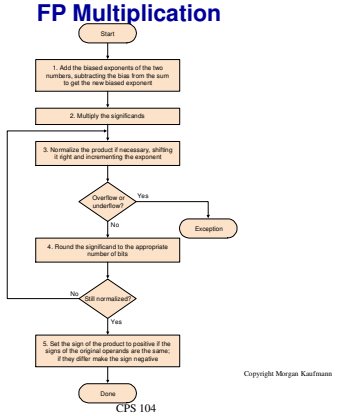
- .5 x -.75 => $1.000 \times 2^{-1} \times 1.100 \times 2^{-2}$

Example FP Multiply

- .5 x -.75 => $1.000 \times 2^{-1} \times 1.100 \times 2^{-1}$
- With Bias $1.000 \times 2^{126} \times 1.100 \times 2^{126}$

1. 126+126-127 = 125
2.
```
  1.000
  1.100
  ----
  0000
 0000
1000
1000
-------
1.100000
```
3. Normalize product: 1.1000000×2^{125}
4. Round: 1.100×2^{125}
5. Compute Sign: different so result is neg
   -1.100×2^{125} (biased) = $-1.100 \times 2^{-2} = -.375$

FP Multiplication



Accuracy

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

- Example:
  $2.56 \times 10^0 + 2.34 \times 10^2$, 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)
  2.34
  0.0256 Guard 5, Round 6
  2.3656
- Round: 2.37×10^0
- Without guard & round bits, result: 2.36×10^0
- 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)

Computer Arithmetic Summary

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

Next

- Review storage elements
- Datapath, Reading Chapter 5…