FP Addition Hardware

What About FP Subtraction?

- Or addition of negative quantities for that matter
  - How to subtract significands that are not in TC form?
  - Can we still use an adder?

- Trick: internally and temporarily convert to TC
  - Add "phantom" –2 in front (–1*2\(^{1}\))
  - Use standard negation trick
  - Add as usual
  - If phantom –2 bit is 1, result is negative
    - Negate it using standard trick again, flip result sign bit
    - Then ignore "phantom" bit (which is now 0 anyway)
  - You’ll want to try this at home!

FP Multiplication

- Assume
  - A represented as bit pattern [S\(_A\), E\(_A\), F\(_A\)]
  - B represented as bit pattern [S\(_B\), E\(_B\), F\(_B\)]
- What is the bit pattern for A*B [S\(_{A*B}\), E\(_{A*B}\), F\(_{A*B}\)]? Pretty much, except for...
  - Normalization
  - Addition of exponents in biased notation (must subtract bias)
  - Tricky: when multiplying two normalized F-bit significands...
  - Where is the binary point?

FP Division

- Assume
  - A represented as bit pattern [S\(_A\), E\(_A\), F\(_A\)]
  - B represented as bit pattern [S\(_B\), E\(_B\), F\(_B\)]
- What is the bit pattern for A/B [S\(_{A/B}\), E\(_{A/B}\), F\(_{A/B}\)]?
- [S\(_{A*B}\), E\(_{A*B}\), F\(_{A*B}\)]? Pretty much, again except for...
  - Normalization
  - Subtraction of exponents in biased notation (must add bias)
  - Binary point placement
  - No need to worry about remainders, either

- A little bit of irony
  - Multiplication/division roughly same complexity for FP and integer
  - Addition/subtraction much more complicated for FP than integer
Accuracy

- Remember our decimal addition example?
  - $9.95 \times 10^1 + 8.00 \times 10^{-1} \rightarrow 1.003 \times 10^2$
  - Extra decimal place caused by de-normalization...
  - But what if our representation only has two digits of precision?
    - What happens to the $3$?
    - Corresponding binary question: what happens to extra $1$s?

- Solution: round
  - Option I: round down (truncate), no hardware necessary
  - Option II: round up (round), need an incrementer
    - Why rounding up called round?
  - Because an extra $1$ is half-way, which is rounded up

More About Accuracy

- Problem with both truncation and rounding
  - They cause errors to accumulate
    - E.g., if always round up, result will gradually "crawl" upwards
- One solution: round to nearest even
  - If un-rounded LSB is $1 \rightarrow$ round up ($011 \rightarrow 10$)
  - If un-rounded LSB is $0 \rightarrow$ round down ($001 \rightarrow 00$)
  - Round up half the time, down other half $\rightarrow$ overall error is stable
- Another solution: multiple intermediate precision bits
  - IEEE 754 defines 3: guard + round + sticky
    - Guard and round are shifted by de-normalization as usual
    - Sticky is $1$ if any shifted out bits are $1$
    - Round up if $101$ or higher, round down if $011$ or lower
    - Round to nearest even if $100$

Numerical Analysis

- Accuracy problems sometimes get bad
  - Addition of big and small numbers
  - Subtraction of big numbers
    - Example, what’s $1 \times 10^3 + 1 \times 10^{-3}$?
      - Intuitively: $1 \times 10^3 = 1$
      - But: $(1 \times 10^3 + 1 \times 10^{-3}) - 1 \times 10^3 = (1 \times 10^3 - 1 \times 10^3) = 0$
- Numerical analysis: field formed around this problem
  - Bounding error of numerical algorithms
  - Re-formulating algorithms in a way that bounds numerical error

One Last Thing About Accuracy

- Suppose you added two numbers and came up with
  - $0101 \; 11111 \; 101$
  - What happens when you round?
    - Number becomes denormalized... arrggggghhh
  - FP adder actually has six steps, not three
    - Align exponents
    - Add/subtract significands
    - Re-normalize
    - Round
    - Potentially re-normalize again
    - Potentially round again
Accuracy, Shmaccuracy?

- Only scientists care? Au contraire
- Intel 486 used equivalent of Modified Booth’s for division
  - Generate multiple quotient bits per step
  - Requires you to guess quotient bits and adjust later
  - Guess taken from a lookup table implemented as PLA

Along came Pentium
- PLA was optimized to return 0 for "impossible" table indices
- Which turned out not to be "impossible" after all
- Result: precision errors in 4th–15th decimal places for some divisors
- "Pentium fdiv bug" is born

Pentium FDIV Bug

- Pentium shipped in August 1994
- Intel actually knew about the bug in July
  - But calculated that delaying the project a month would cost ~$1M
  - And that in reality only a dozen or so people would encounter it
- They were right... but one of them took the story to EE Times
- By November 1994, firestorm was full on
  - IBM said that typical Excel user would encounter bug every month
  - Assumed 5K divisions per second around the clock
  - People believed the story
  - IBM stopped shipping Pentium PCs
- By December 1994, Intel promises full recall
  - Total cost: ~$550M
  - All for a bug which in reality maybe affected a dozen people

Summary of Floating Point

- FP representation
  - S*F*2^E
  - IEEE754 standard
  - Representing fractions
  - Normalized numbers
- FP operations
  - Addition/subtraction: hard
  - Multiplication/division: logarithmic no harder than integer
- Accuracy problems
  - Rounding and truncation
- Upshot: FP hardware is tough
  - Thank lucky stars that ECE 152 project has no FP

Unit Recap: Arithmetic and ALU Design

- Integer Arithmetic and ALU
  - Binary number representations
  - Addition and subtraction
  - The integer ALU
  - Shifting and rotating
  - Multiplication
  - Division
- Floating Point Arithmetic
  - Binary number representations
  - FP arithmetic
  - Accuracy