Topic V24

Floating-Point Arithmetic

Reading: (Section 3.5)

Floating Point Addition (Decimal)

How do we perform the following addition (assume you can only have four decimal digits)?

$$9.999_{10} \times 10^{1} + 1.610_{10} \times 10^{-1}$$

Step 1: Align decimal point of the number with smaller exponent (notice loss of precision)

$$9.999_{10} \times 10^{1} + 0.016_{10} \times 10^{1}$$

Step 2: Add significands

$$9.999_{10} \times 10^{1} + 0.016_{10} \times 10^{1} = 10.015_{10} \times 10^{1}$$

Step 3: Renormalize the result

$$10.015 \times 10^{1} = 1.0015 \times 10^{2}$$

Step 4: Round-off the result to the representation available

$$1.0015 \times 10^{2} = 1.002 \times 10^{2}$$

Floating Point Addition (example)

Convert the numbers 0.5_{10} and -0.4375_{10} to floating point binary representation, and then perform the binary floating-point addition of these numbers

$$\begin{array}{r} 0.4375 \\ \underline{-0.2500} \\ 0.1875 \\ \underline{-0.1250} \\ 0.0625 \\ \underline{-0.0625} \\ 0.0000 \end{array}$$

$0.5_{10} = 2^{-1} = 0.1_2$ $\boxed{1.000_2 \times 2^{-1}}$

$$\begin{aligned} 0.4375_{10} &= 0.25_{10} + 0.125_{10} + 0.0625_{10} \\ &= 2^{-2} + 2^{-3} + 2^{-4} \\ &= 0.0111_2 \times 2^0 \end{aligned}$$

$\boxed{1.110_2 \times 2^{-2}}$

$2^{-1}=0.5$
$2^{-2}=0.25$
$2^{-3}=0.125$
$2^{-4}=0.0625$
$2^{-5}=0.03125$

Which number should have its significand adjusted?

$$\begin{aligned} 0.5_{10} - 0.4375_{10} &= 1.000_2 \times 2^{-1} - 0.111_2 \times 2^{-1} \\ &= 0.001_2 \times 2^{-1} \\ &= 1.000_2 \times 2^{-4} \end{aligned}$$

Floating Point Multiplication (Decimal)

Assume that we only can store four digits of the significand and two digits of the exponent in a decimal floating-point representation.

How would you multiply $1.110_{10} \times 10^{10}$ by $9.200_{10} \times 10^{-5}$ in this representation?

Step 1: Add the exponents new exponent = 10 – 5 = 5

Step 2: Multiply the significands

Step 3: Normalize the product

$$10.212_{10} \times 10^{5} = 1.0212_{10} \times 10^{6}$$

Step 4: Round off the product

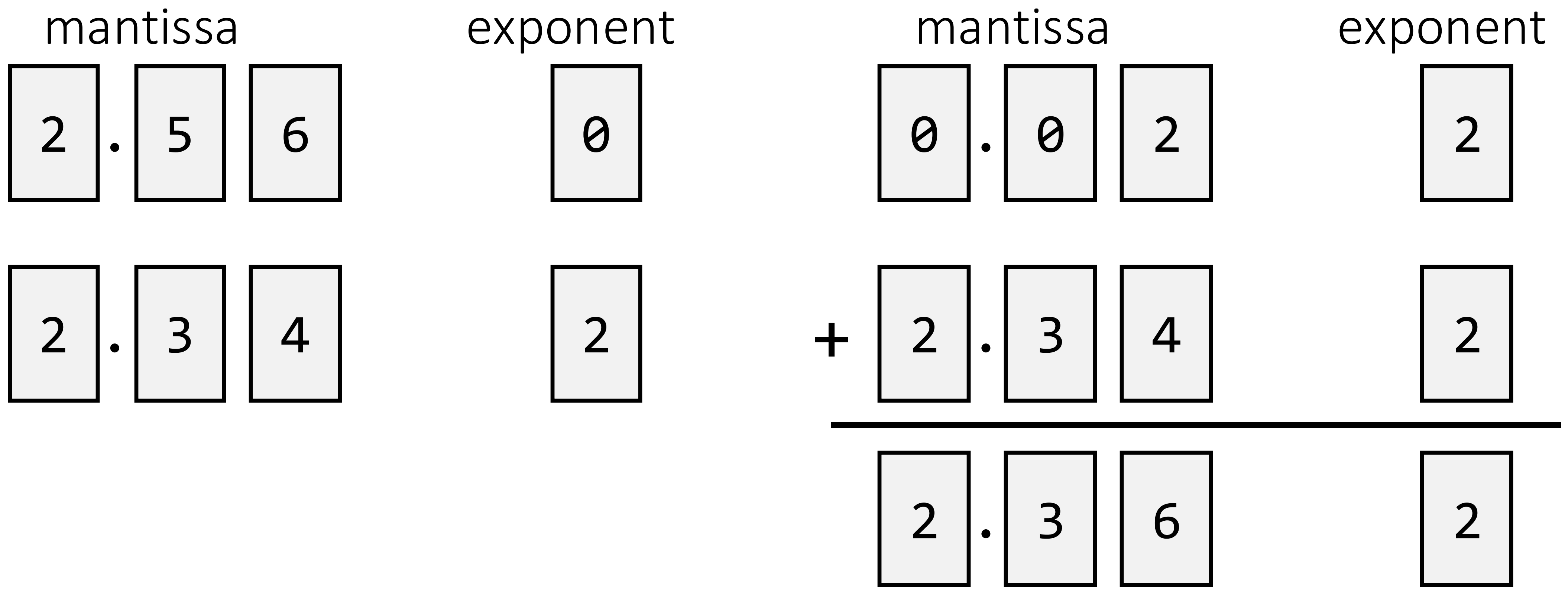
$$1.0212_{10} \times 10^{6} = 1.021_{10} \times 10^{6}$$

```
     1.110
   x 9.200
   -------
      0000
     0000
    2220
   9990
   -------
  10.212000
```

Accurate Arithmetic

Assume that you can only store three significant decimal digits

Add $2.56_{10} \times 10^0$ to $2.34_{10} \times 10^2$



Accurate Arithmetic

Now assume that you also have room for a guard digit and for a round digit

Add $2.56_{10} \times 10^0$ to $2.34_{10} \times 10^2$

mantissa	exponent
2 . 5 6	0
2 . 3 4	2

	mantissa	guard	round	exponent
	0 . 0 2	5	6	2
+	2 . 3 4	0	0	2
	2 . 3 6	5	6	2

We round 00-49 down, and 51-99 up

We round 50 to the nearest even number

Thus the result of the sum is 2.37

Rounding

Decimal	Binary	
0.00	0.00	0.00
0.01	0.01	0.25
0.02	0.10	0.50
…	0.11	0.75
0.49		
0.50		
0.51		
…		
0.97		
0.98		
0.99		

Sticky Bit

Ideally, we would like to do the intermediate computation with infinite precision and round the result

The problem is how to differentiate between these two numbers:

$0.50\ldots00_{10}$

$0.50\ldots01_{10}$

The solution is to use a sticky bit that is set whenever there are non-zero bits to the right of the round bit

Example with no Sticky Bit

$5.01_{10} \times 10^{-1}$ to $2.34_{10} \times 10^{2}$ with guard and round digits but no sticky bit

mantissa			exponent
5.	0	1	-1
2.	3	4	2

	mantissa			guard	round	exponent
	0.	0	0	5	0	2
+	2.	3	4	0	0	2
	2.	3	4	5	0	2

We round 50 to the nearest even

Thus the result is 2.34

Example with Sticky Bit

$5.01_{10} \times 10^{-1}$ to $2.34_{10} \times 10^{2}$ with guard and round digits and a sticky bit

mantissa			exponent		mantissa			guard	round	sticky	exponent
5.	0	1	-1		0.	0	0	5	0	1	2
2.	3	4	2	+	2.	3	4	0	0	0	2
					2.	3	4	5	0	1	2

With sticky bit equal 1, 50 is rounded up

Thus the result is 2.35

Accurate Arithmetic

IEEE Std 754 specifies additional rounding control

- Extra bits of precision (guard, round, sticky)
- Choice of rounding modes
- Allows programmer to fine-tune numerical behaviour of a computation

Not all FP units implement all options

- Most programming languages and FP libraries just use defaults

Trade-off between hardware complexity, performance, and market requirements

The importance of rounding

The Vancouver Stock Exchange's Rounding Error

Date: 1982

A minor decimal distinction with a major cost

```
- return floor(value)
+ return round(value)
```

This is an approximation of the code that the Vancouver Stock Exchange would have used.

Chase Felker/Slate

In early 1982, the Vancouver Stock Exchange unveiled an electronic stock index initially pegged to a value of 1,000 points. In two years it dropped to half its original value—a confusing trend amid the bull market of the early 1980s. An investigation revealed that the calculations of the index were wrong in just one command, using floor() rather than round(). This command meant that instead of rounding to the third decimal place, the value was being truncated. (Digital computers necessarily have finite resolution, which necessitates rounding or truncation.) So if the index was calculated as 532.7528, it was stored as 532.752, rather than rounded up to 532.753. Over the course of thousands of calculations a day, this seemingly minor difference—essentially rounding down every single time—amounted to a dramatic loss in value. The programming mistake was finally fixed in November 1983, after the index closed around 500 on a Friday. It reopened on Monday at over 1,000, its lost value restored. *—Lav Varshney, assistant professor, University of Illinois at Urbana-Champaign*

https://slate.com/technology/2019/10/consequential-computer-code-software-history.html?fbclid=IwAR34FuObp93Q2wHQT60SkQxfJvcbaGHIac4uE7EhIxDlnDqXoVqLlC_AM8A

Associativity of FP Addition

Assume that:

$x = -1.5_{10} \times 10^{38}$

$y = 1.5_{10} \times 10^{38}$

$z = 1.0_{10}$

Thus, floating point addition is <u>not associative</u>

Compute x+(y+z) and (x+y)+z

Assume that we only have room to store 32 decimal digits for the significand of any number

$x + (y + z) = -1.5_{10} \times 10^{38} + (1.5_{10} \times 10^{38} + 1.0_{10})$

$= -1.5_{10} \times 10^{38} + (1.5_{10} \times 10^{38} + 0.0_{10} \times 10^{38})$

$= -1.5_{10} \times 10^{38} + 1.5_{10} \times 10^{38}$

$= 0.0_{10}$

$(x + y) + z = (-1.5_{10} \times 10^{38} + 1.5_{10} \times 10^{38}) + 1.0_{10}$

$= 0.0_{10} + 1.0_{10}$

$= 1.0_{10}$

Associativity

Parallel programs may interleave operations in unexpected orders

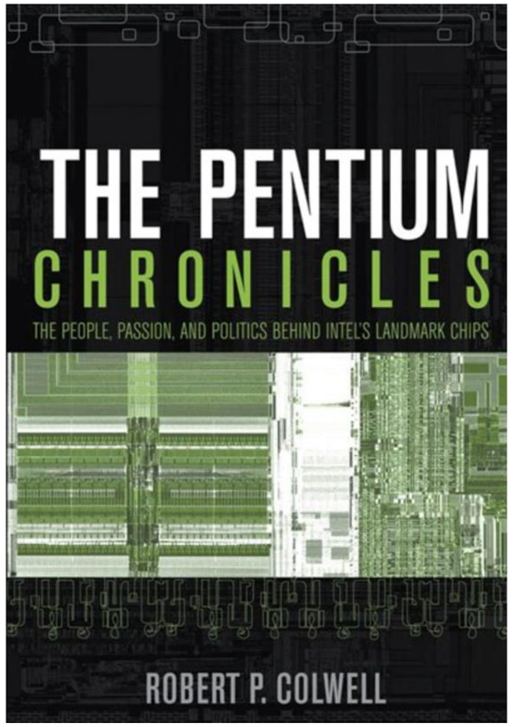
Assumptions of associativity may fail

Need to validate parallel programs under varying degrees of parallelism

Who Cares About FP Accuracy?

- Important for scientific code
-   But for everyday consumer use?
-     “My bank balance is out by 0.0002¢!”

- The Intel Pentium FDIV bug
-   The market expects accuracy
-   See Colwell, The Pentium Chronicles


THE PENTIUM
CHRONICLES
THE PEOPLE, PASSION, AND POLITICS BEHIND INTEL'S LANDMARK CHIPS
ROBERT P. COLWELL

Los Angeles Times

Mathematician Finds Intel's Pentium Doesn't Compute : Technology: A flaw that the company failed to disclose in June causes errors in complex calculations.

BY MARTHA GROVES

NOV. 24, 1994 12 AM PT

TIMES STAFF WRITER

SAN FRANCISCO — A mathematician in Lynchburg, Va., was the first to get bugged by a flaw in Intel Corp.'s popular Pentium chip.

While doing complex division problems, Thomas R. Nicely, a mathematics professor at Lynchburg College, detected a defect that caused his department's three Pentium-based computers to spew out faulty numbers in the ninth place to the right of the decimal point and beyond. So he called Intel a month ago, and now he's garnering a Warholesque modicum of fame as the man who blew the whistle on the chink in

Floating Point in ISA

ISAs support arithmetic

- Signed and unsigned integers
- Floating point is an approximation of real numbers

Bounded range and precision

- Operations can overflow and underflow

RISC-V ISA

- 20 most popular instructions account for 76% of all instructions executed for the SPEC CPU2006
  - FP instructions account for 16.4% of this