Computer Architecture
(Floating Point Arithmetic)

विद्याधनं सर्वधन प्रधानम्

Subhasis Bhattacharjee
Department of Computer Science and Engineering,
Indian Institute of Technology, Jammu

September 18, 2023

Outline

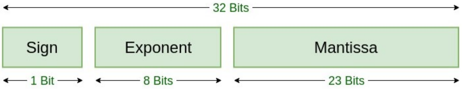
1. Floating Point Representation - Review
2. Floating Point Arithmetic
3. Floating Point Addition
4. Floating Point Multiplication
5. Floating Point Division
6. Floating Point Division Using Goldschmidt Division

Floating Point Representation - Review

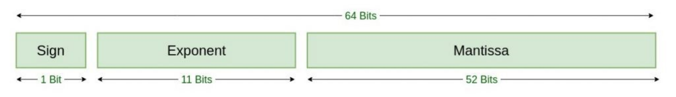
Floating Point Number System - Review

- A number representation specifies some way of encoding a number, usually as a string of digits.
- floating-point number represents real numbers approximately, using an integer with a fixed precision, called the significand, scaled by an integer exponent.
- IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point number format and arithmetic established in 1985.

Single Precision Format (32 bits)



Double Precision Format (64 bits)



IEEE 754 Floating Point Number provides **Two Forms**

- Normalized number or (Normal) form
- Denormalized numbers or (denormal) form

Normalised form of a 32 bit (normal) floating point number

$$A = (-1)^S \times P \times 2^{E-bias}, \ (1 \leq P < 2, E \in \mathbb{Z}, 1 \leq E \leq 254)$$

Normalised form of a 32 bit (denormal) floating point number

$$A = (-1)^S \times P \times 2^{-126}, \ (0 \leq P < 1)$$

Symbol	Meaning
S	Sign bit (0(+ve), 1(-ve))
P	Significand (form: 1.xxx(normal) or 0.xxx(denormal))
M	Mantissa (fractional part of significand)
E	(exponent + 127(bias))
$\mathbb{Z}$	Set of integers

Floating Point Arithmetic

Floating Point Arithmetic - Scope of our discussion

- We will follow IEEE 754 Single Precision (32 bits) format only.
- We will discuss 4 arithmetic operations (add, subtract, multiply and division)
- We will mostly consider only the normal form
- The algorithms / procedures are similar for denormal form (minute differences).
- Various components of a floating point number are packed into 32-bits
- Generalized Idea for each arithmetic operation:
  1. Unpack various components from 32-bits format
  2. Do some computations on each component (depending on the operation)
  3. Rounding (if any)
  4. Pack the result into 32-bits format
  5. ReNormalize the intermediate values or end result - if require.
- We will also study:
  - Rounding methods
  - ReNormalization process

Floating Point Addition

Addition of Two Real Numbers - TODO

- Suppose we want to add 123.45 and 3.24
- First we align the decimal points.

Add : $A + B$

- Unpack the E fields
  - Let E_A, E_B are E fields of A and B, respectively.
  - Let the E field of the result be $\rightarrow E_C$.
- Unpack the significand (P)
  - P contains $\rightarrow$ 1 bit before the decimal point, 23 mantissa bits (24 bits)
  - Unpack to a 25 bit number (unsigned)
    - W $\rightarrow$ Add a leading 0 bit, 24 bits of the signficand
    - **Why do we add a leading 0 bit?** - If we add two numbers it may generate an extra bit.
  - Let significands of A and B be P_A and P_B

Addition - Algorithm

- With no loss of generality Assume $E_A >= E_B$
- Let us initially set $W \leftarrow$ unpack (P_B)
- We make their exponents equal and shift W to the right by $(E_A - E_B)$ positions [Let $\gg$ denotes the right-shift operation]

Addition Algorithm

$W = W \gg (E_A - E_B)$
$W = W + P_A$

Renormalisation

- Let the significand represented by register, W, be P_W
- There is a possibility that $P_W >= 2$
- In this case, we need to renormalise
  1. $W \leftarrow W \gg 1$
  2. $E_A \leftarrow E_A + 1$
- The final result
  - Sign bit (same as sign of A or B)
  - Significand (P_W), exponent field (E_A)

Addition Example - 1

Add the numbers: $(1.01)_2 * 2^3 + (1.11)_2 * 2^1$

Answer: Step-by-step computations:

1. $A = 1.01 * 2^3$ and $B = 1.11 * 2^1$
2. $W = 01.11$ (significand of B)
3. $E = 3$, Number of Bits to shift = (3-1) = 2
4. $W = 01.11 \gg 2 = 00.0111$
5. $W + P_A = 00.0111 + 01.0100 = 01.1011$
6. Result: $C = 1.011 * 2^3$

The decimal point in W is shown for enhancing readability. For simplicity, biased notation is not used.

Addition Example - 2

Add : $(1.01)_2 * 2^3 + (1.11)_2 * 2^2$

Answer: Step-by-step computations:

1. $A = 1.01 * 2^3$ and $B = 1.11 * 2^2$
2. $W = 01.11$ (significand of B)
3. $E = 3$, Number of Bits to shift $= (3 - 2) = \mathbf{1}$
4. $W = 01.11 \gg \mathbf{1} = 00.111$
5. $W + P_A = 00.111 + 01.0100 = 10.001$
6. Normalisation: $W = 10.001 \gg 1 = 1.0001$, $E = 4$
7. Result: $C = 1.0001 * 2^4$

Rounding

- Assume that we were allowed only two mantissa bits in the previous example
- We need to perform rounding
- Consider the sum(W) of the significands after we have normalised the result
- $W \leftarrow (P + R) * 2^{-23}$, $(R < 1)$
- P represents the significand of the temporary result
- R (is a residue)

Rounding Objectives

Rounding Goal

- Modify P to take into account the value of R
- Then, discard R
- Process of rounding : P $\rightarrow$ P'

IEEE 754 Rounding Modes

Truncation

- P' = P
- Example in decimal: 9.5 $\rightarrow$ 9, 9.6 $\rightarrow$ 9

Round to $+\infty$

- P' = $\lceil$ P + R $\rceil$
- Example in decimal: 9.5 $\rightarrow$ 10, -3.2 $\rightarrow$ -3

Round to $-\infty$

- P' = $\lfloor$ P + R $\rfloor$
- Example in decimal: 9.5 $\rightarrow$ 9, -3.2 $\rightarrow$ -4

IEEE 754 Rounding - II

Round to nearest

- P' = [P + R]
- Example in decimal :
- 9.4 $\rightarrow$ 9 , 9.5 $\rightarrow$ 10 (even)
- 9.6 $\rightarrow$ 10 , -2.3 $\rightarrow$ -2
- -3.5 $\rightarrow$ -4 (even)

Rounding Modes – Summary - **TODO Overfull hbox**

Notation: $\wedge$ (logical AND), R (residue), P (significand)		
Rounding Mode	**Condition for incrementing the significand**	
	Sign of the result (+ve)	**Sign of the result (-ve)**
Truncation		
Round to $+\infty$	$R > 0$	
Round to $-\infty$		$R > 0$
Round to Nearest	$(R > 0.5) \| \| (R = 0.5 \wedge lsb(P) = 1)$	$(R > 0.5) \| \| (R = 0.5 \wedge lsb(P) = 1)$

Implementing Rounding

We need three bits for Implementation

- lsb(P)
- msb of the residue (R) $\rightarrow$ r (round bit)
- OR of the rest of the bits of the residue (R) $\rightarrow$ s (sticky bit)

Notation: $\wedge$ (logical AND), $\vee$ (logical OR), r (round bit), s(sticky bit)	
Condition on Residue	Implementation
$R > 0$	$r \vee s = 1$
$R = 0.5$	$r \wedge s = 1$
$R > 0.5$	$r \wedge s = 1$

Renormalisation after Rounding

- In rounding : we might increment the significand
- We might need to renormalise
- After renormalisation
- Possible that E becomes equal to 255
- In this, case declare an overflow16

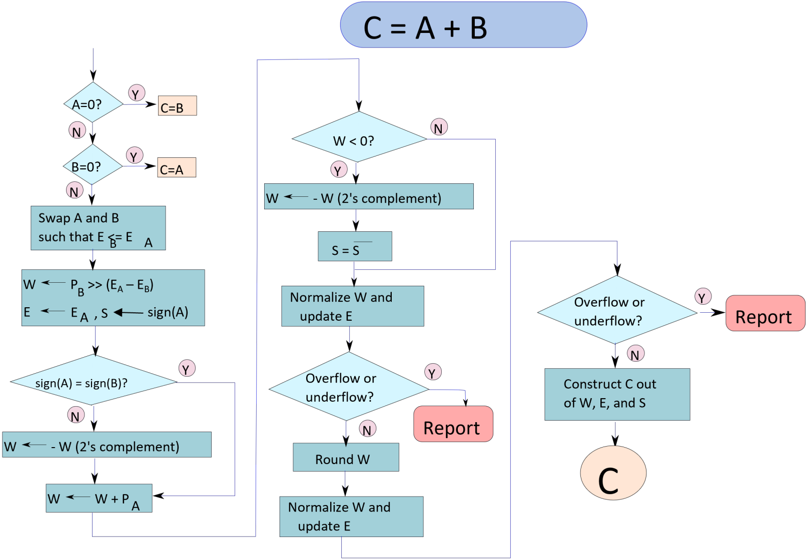
Addition of Numbers (Opposite Signs)

$C = A + B$

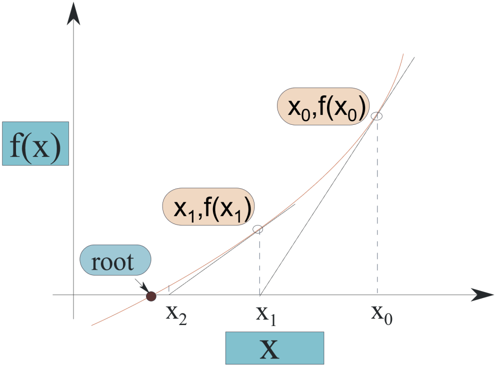
Same assumption $E_A >= E_B$. Step-by-step computations:

1. Load W with the significand of B (P_B)
2. Take the 2's complement of W (W = -B)
3. $W \leftarrow W \gg (E_A - E_B)$
4. $W \leftarrow W + P_A$
5. If ($W < 0$) replace it with its 2's complement. Flip the sign of the result.

- Normalise the result
  - Possible that $W < 1$
  - In this case, **keep shifting W** to the left till it is in normal form. (simultaneously decrement E_A)
- Round and Renormalise


C = A + B
A=0?
Y
C=B
N
B=0?
Y
C=A
N
Swap A and B
such that E B <= E A
W ← P B >> (E A − E B)
E ← E A , S ← sign(A)
sign(A) = sign(B)?
Y
N
W ← - W (2's complement)
W ← W + P A
W < 0?
N
Y
W ← - W (2's complement)
S = S
Normalize W and
update E
Overflow or
underflow?
Y
Report
N
Round W
Normalize W and
update E
Overflow or
underflow?
Y
Report
N
Construct C out
of W, E, and S
C

Newton Raphson Method - Pictorial



Floating Point Multiplication

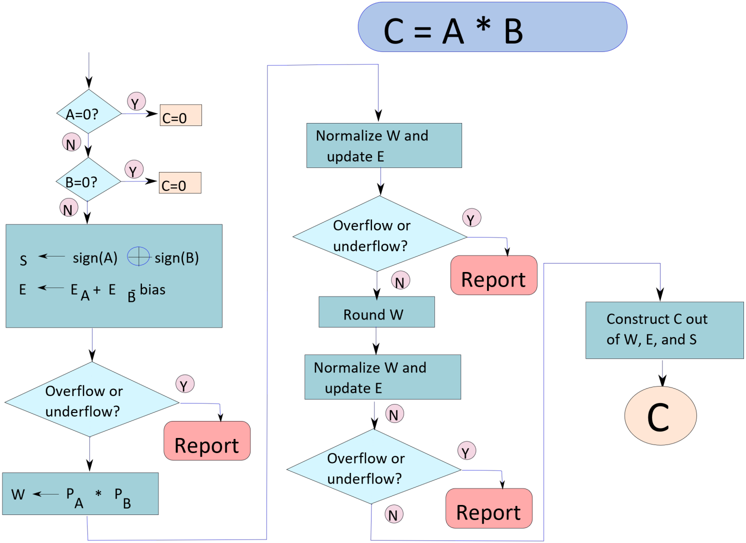
Multiplication of Two FP Numbers

Multiplication: $C = A * B$

Step-by-step computations:

1. $E \leftarrow E_A + E_B$ - bias
2. $W \leftarrow P_A * P_B$
3. Normalise (shift left or shift right)
4. Round
5. Renormalise

Flowchart of Multiplication of Two FP Numbers



Multiplication Example - 1 (Same sign)

Multiplication Example - 2 (Opposite sign)

Multiplication Example - 3

3 Examples TODO

Case: 1st Normalize

Case: Normalize, Round

Case: Normalize, Round, Renormalize

Floating Point Division

Floating Point Division

Few points on FP Division

- Divide A/B to produce C
- There is no notion of a remainder in FP division

Simple Division Algorithm

1. $E \leftarrow E_A - E_B + \text{bias}$
2. $W \leftarrow P_A \; / \; P_B$
3. normalise, round, renormalise (as needed).

Complexity : $O(n \log(n))$

Floating Point Division Using Goldschmidt Division

Goldschmidt Division

Division: C = A / B

Step-by-step computations:

1. Compute the B' = reciprocal of B (i.e., 1/B)
2. Using the standard floating point multiplication algorithm compute A * (1/B)

Computing (1/B) - the reciprocal of B

- Ignoring the exponent.
- Let us compute $(1/P_B)$
- If B is a normal floating point number
  - $1 <= P_B < 2$
  - $P_B = 1 + X$, where $(X < 1)$

$$\begin{aligned}
\frac{1}{P_{\mathrm{B}}} &= \frac{1}{1+X}, (P_{\mathrm{B}} = 1+X, 0 < X < 1) \\
&= \frac{1}{1+1-X'}, (X' = 1-X, X' < 1) \\
&= \frac{1}{2-X'} \\
&= \frac{1}{2} * \frac{1}{1-\frac{X'}{2}} \\
&= \frac{1}{2} * \frac{1}{1-Y}, (Y = \frac{X'}{2}, Y < \frac{1}{2})
\end{aligned} \tag{1}$$

Computing Reciprocal - Page 2 - $\frac{1}{1-Y}$

$$
\begin{aligned}
\frac{1}{1-Y} &= \frac{1+Y}{1-Y^2} \\
&= \frac{(1+Y)(1+Y^2)}{1-Y^4} \\
&= \frac{(1+Y)(1+Y^2)(1+Y^4)}{1-Y^8} \\
&= \cdots \\
&= \frac{(1+Y)(1+Y^2)(1+Y^4)\cdots(1+Y^{16})}{1-Y^{32}} \\
&\approx (1+Y)(1+Y^2)(1+Y^4)\cdots(1+Y^{16})
\end{aligned} \tag{2}
$$

How long to continue this series expansion

- No point considering Y^{32}. It needs 31 leading 0's.
- Cannot be represented in IEEE 754 format (mantissa contains only 23 bits).

Generating $\frac{1}{1-Y}$

$$\frac{1}{1-Y} = (1+Y)(1+Y^2)(1+Y^4)\cdots(1+Y^{16}) \tag{3}$$

How to compute this long series - Using FP Multiplier

- We can compute Y^2 using a FP multiplier.
- Again square it to obtain Y^4, Y^8, and Y^{16}
- We Need 4 multiplications, and 5 additions, to generate all the terms
- Need 4 more multiplications to generate the final result $\frac{1}{1-Y}$.

Computing $\frac{1}{P_B}$

Recall derivation of $\frac{1}{P_B}$

$$\frac{1}{P_{\text{B}}} = \frac{1}{2} * \frac{1}{1 - Y} \tag{4}$$

- A single Right shift (i.e., 1 bit right-shift) of $\frac{1}{1-Y}$
- $\frac{1}{P_B} \leftarrow \frac{1}{1-Y} >> 1$

GoldSchmidt Division Summary

- Time complexity of finding the **reciprocal** $O(\log(n))^2$
- Time required for all the **multiplications and additions** $O(\log(n))^2$
- Total Time : $O(\log(n))^2$