भारतीय प्रौद्योगिकी संस्थान तिरुपति
TIRUPATI

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
Department of Electrical Engineering
VLSI for Signal Processing course project report
Ambati Sathvik EE17B002, Tirunagari Rahul EE17B030

Operations in IEEE Standard 754 Floating Point Representation

Guide: Dr.Vikramkumar Pudi

Abstract

Basic math operations addition and multiplication are carried out using IEEE standard 754 Floating Point representation. Advantages of the representation, its necessity, scheme, range, and exceptions are described.

Introduction

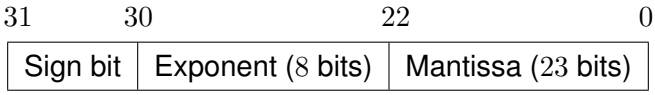
The standard representation of binary numbers, including fractions, is required to perform operations such as addition, multiplication, and division. Fixed point representation freezes the binary point location, thereby storing integral parts and fractional parts, making the addition operation simpler. Variable-length representation groups and encodes bits into 7-bit groups and minimises the storage. Rational representation holds the numerator and denominator separately[2].

Floating-point representation does not allocate a predetermined number of bits for integral and fractional parts (i.e., the binary point is allowed to float). A number is represented with its fixed-point number (mantissa), and exponent conveys the binary point location. Mantissa decides the precision, and the exponent bits are responsible for the range. Several standards such as VAX, IBM 360, Cyber, Cray I, IEEE 754 were proposed to represent the numbers in this format[3][5]. The IEEE standard handled the trade-off between accuracy and dynamic range fairly.

Method

Single precision IEEE Standard 754 Floating Point representation uses 32 bits to represent numbers between $\pm 10^{-44.85}$ to $10^{38.53}$[1].

A number N is expressed in the form of $\pm 1.m_1 m_2 .. m_{23}$ x $2^{e_1 e_2 .. e_8}$, where m_i, $1 \leq i \leq 23$ are the mantissa bits, e_i, $1 \leq i \leq 8$ are exponent bits. The number N is represented with these 32-bits, including the sign bit[4].



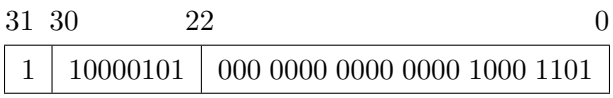
- Sign information is provided by the MSB, it is 0 for positive numbers and 1 for negative numbers.
- The following 8-bits are dedicated for exponent. The exponent is negative while representing some of the integral part in mantissa. To avoid the signed representation for the exponent, 127 (bias) is added to the actual exponent so that the same range is now unsigned.
- Mantissa occupies the latter 23 bits to represent the fraction part which is followed by 1 and binary point of the normalised form.

Example: Representing -81.625 in the discussed format.

Binary representation is -1010001.101

$$= -1.010001101 \text{ x } 2^6$$

biased exponent is $6 + 127 = 133 = \underline{10000101}$



Addition operation:

Procedure

- Exponents of both the numbers should be identified and the difference (d) is noted.
- Mantissa of the smaller number is shifted right by d bits and both the input exponents are made same as the larger one.
- Exponent of the output is also the same as higher exponent
- Mantissa of the output number is the sum of input mantissas.
- Output sign is obtained after adding the mantissas.

Precautions

- Sign bit affects only the mantissa, so it is converted into $2's$ complement for negative numbers.
- Shifting the mantissa directly refers to shift of the binary point. So the incoming bits during right shift of mantissa will be 1 followed by zeroes as the number is actually represented as $1.mantissa$ x 2^{exp}.
- Summation of 23-bit mantissas result in 24-bit number ($M_A + M_B$). MSB of the sum suggests the correction in output exponent, however, truncation / rounding is performed only if the input numbers are of same sign.
- If the numbers are of opposite sign, the output sign is defined by bit 23 (from LSB) of $M_A + M_B$.
- For negative output numbers, the mantissa is converted back to its positive value.
- Overflow occurs if the numbers are of same sign and if one of the exponent is close to ± 127.

Example: Addition of A 0 | $0111\ \ 1110$ | $0110\ \ 1111\ \ 0000\ \ 1111\ \ 0000\ \ 011$

and B 0 | $1000\ \ 0000$ | $0000\ \ 1111\ \ 0000\ \ 1111\ \ 0000\ \ 100$.

$d = 128 - 126, \Rightarrow$ mantissa of A $>> 2$.

Modified A is 0 | $1000\ \ 0000$ | **$\underline{01}$**$01\ \ 1011\ \ 1100\ \ 0011\ \ 1100\ \ 000$

Mantissa_C is $0101\ \ 1011\ \ 1100\ \ 0011\ \ 1100\ \ 000$

$$+\ 0000\ \ 1111\ \ 0000\ \ 1111\ \ 0000\ \ 100$$

$$0\ \ \underline{0}110\ \ 1010\ \ 1101\ \ 0010\ \ 1100\ \ 100$$

Bit 23 (from LSB) indicates the overall sum is positive.

Binary representation of C is $\underline{0}$ | $1000\ \ 0000$ | $\underline{0}110\ \ 1010\ \ 1101\ \ 0010\ \ 1100\ \ 100$.

A is 1.43382298946380615234 x 2^{-1} = 0.71691149473190307617

B is 1.05882298946380615234 x 2^{1} = 2.11764597892761230468

C is 1.41727876663208007813 x 2^{1} = 2.83455753326416015626

The error after addition is 5.96 x $10^{-}8$, due to the truncation.

for B = 1 | $0111\ \ 1110$ | $0110\ \ 1111\ \ 0000\ \ 1111\ \ 0000\ \ 011$ = $-$A

Modified B is 0 | $0111\ \ 1110$ | $1001\ \ 0000\ \ 1111\ \ 0000\ \ 1111\ \ 101$ ($2s$ complement)

Mantissa_C is $0110\ \ 1111\ \ 0000\ \ 1111\ \ 0000\ \ 011$

$$+\ 1001\ \ 0000\ \ 1111\ \ 0000\ \ 1111\ \ 101$$

$$1\ \ \underline{0}000\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 000$$

Now, representation of C is $\underline{0}$ | $0111\ \ 1110$ | $\underline{0}000\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 000$.

Multiplication:

Procedure

- Exponents of both the numbers should be unbiased, and the sum (s) is biased back.
- Both the mantissas with prefix 1 are directly multiplied (because the multiplication of normalised numbers is actually performed as $1.abc..$ and $1.xyz...$). The resultant number m of $(23+1)*2$ bits are truncated / rounded off such that most significant 23 bits are retained for the output mantissa.
- Output sign bit can be directly calculated from the input signs.
- The possible exponents of output, s or $s+1$ is decided by the MSB of m.

Precautions

- Output should be normalised, based on the most significant bits of m.
- Rounding off the extra bits should be preferred to the truncation.
- Sum of the exponents should be in range $[-127, 128]$.

Example: Multiplication of A $(126.13)_{10}$ = 0 | $1000\ \ 0101$ | $1111\ \ 1000\ \ 1000\ \ 0101\ \ 0001\ \ 111$ and B $(26.1)_{10}$ = 0 | $1000\ \ 0011$ | $1010\ \ 0001\ \ 1001\ \ 1001\ \ 1001\ \ 101$.

Sign of C is obtained from Sign A $\oplus$ Sign B $(0 \oplus 0 = 0)$

Exponents of A and B sums up to give exponent of C

$(133 - 127 + 131 - 127 + 127 = 137)$.

Mantissas of A and B are multiplied with prefix $1.$ to give,

$1.1111\ \ 1000\ \ 1000\ \ 0101\ \ 0001\ \ 111$ x $1.1010\ \ 0001\ \ 1001\ \ 1001\ \ 1001\ \ 101$

= $11.00110110111111111000110011101101000000010000011$

Truncation of the normalised product to 23 bits give

$1.1001\ \ 1011\ \ 0111\ \ 1111\ \ 1100\ \ 011$ x 2^1

The rise in exponent is added to the pre-computed exponent of C$(137 + 1 = 138)$. So, AxB is 0 | $1000\ \ 1010$ | $1001\ \ 1011\ \ 0111\ \ 1111\ \ 1100\ \ 011$ with $8x10^{-5}$ error. The error occurs due to truncation and imprecise representation of input numbers.

References

[1] Ayusharma. *IEEE Standard 754 Floating Point Numbers*. https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/?ref=rp. 2020.

[2] Douglas F Elliott. *Handbook of digital signal processing: engineering applications*. Elsevier, 2013.

[3] William Kahan. "Why do we need a floating-point arithmetic standard?" In: *Whitepaper: Online: http://www. eecs. berkeley. e du/˜ wkahan/ieee754status/why-ieee. pdf* (1981).

[4] Tutorials point. *Fixed Point and Floating Point Number Representations*. https://www.tutorialspoint.com/fixed-point-and-floating-point-number-representations. 2020.

[5] Ralph. *Variable length integers*. https://golb.hplar.ch/2019/06/variable-length-int-java.html. 2019.