Introduction to Floating point calculations and IEEE 754 standard

Jamil Khatib

August 10, 2000

1 Introduction

Floating Point is a representation of real (fractional) numbers. In this representation the location of the fractional point can be moved from one location to another according to the precision. "thats where the name came from". It takes the general format as
Exp. 0. Fraction

Although these numbers solve many problems of integers, it has its own problems and considerations. This is because the resources available for storing these numbers are limited and so we can not represent infinite number of digits and not like real life numbers and calculations. Rounding method and order of calculations are few considerations of floating number calculations.

2 Introduction to IEEE-754 standard

In the early days of computers, vendors start developing their own representations and methods of calculations. These different approaches lead to different results in calculations. So the IEEE organization defined in the IEEE-754 standard a representation of the floating point numbers and the operations.

3 Representation

- **Representation** As in all floating point representations, the IEEE representation divides the number of bits into three groups, the exponent and the fractional part.

- **Fractional numbers** are represented as sign-magnitude which needs a reserved bit for the sign.

- **The exponent** is based on the biased representation. Which means if k is the value of the exponent bits, then the exponent of the floating point number is $k-$the bias. So to represent the exponent *zero* the bits should hold the value of the bias.

- **Hidden-bit** Another feature of the IEEE representation the is the hidden bit. This bit is the only bit to the left of the fraction point. this bit is assumed to be 1 which gives an extra bit of storage in the representation in increases the precision.

	Single	Single-Extended	Double	Double-Extended	Quad-Precision
Exponent(max)	+127	1023	+1023	+16383	+16383
Exponent(min)	-126	1022	-1022	-16382	-16382
Exponent Bias	+127	+1023	+1023	+16383	+16383
Precision(#bits)	24	$\geq$32	53	$\geq$64	113
Total Bits	32	$\geq$43	64	80	128
Sign bits	1	1	1	1	1
Exp Bits	8	11	11	15	15
Fraction	23	$\geq$32	52	64	112

4 Precision

- **Precisions** The IEEE-754 defines set of precisions which depends on the number of bits used. There are two main precisions, the single and the double "the quad-precision is not used often".

- **Extended** The standard also define an extended precision for both standard precisions. The number of used bits is enlarged. The standard defines the minimum number of bits of the extended format. and its up to the implementer to increase it. The IEEE standard *requires* that the implementation should support the corresponding extended format.

- **Reason** The main reason for extended format came from calculators which displays 10 digits but use 13 digits internally, which makes the user feel as if the calculator computes to 10 digits accuracy[1]. This feature is important to make all calculations on all IEEE-754 platforms give the same results after rounding. It is also needed to distinguish between exact and inexact results.

-

- **Note:** The standard requires that all calculations are made in extended format and then rounded to the precision. It is important to round the extended result of each operation alone to the corresponding precision. Because if it is not done the final result will depend on the extra bits and produce and unexpected results.

5 Normalization

- Normalization is the act of shifting the fractional part in order to make the left bit of the fractional point is one. During this shift the exponent is incremented.
- **Normalized numbers** are the numbers that have their MSB 1 is in the most left bit of the fractional part.
- **Denormalized numbers** are the opposite of the normalized numbers. (i.e. the MSB 1 is not in the most left bit of the fractional part).
- **Operations:** Some operations requires that the exponent field is the same for all operands (like addition). In this case one of the operands should be denormalized.
- **Importance:** Denormalized numbers have important use in some operations and numbers. For example[1], assume minimum exponent is -98, and number of digits is 3 and to perform the operation $x - y$ where $x = 6.87 \times 10^{-97}$ and $y = 6.81 \times 10^{-97}$. The result of this operation is 0.06×10^{-97} if This number is normalized in decimal it will be 6.00×10^{-99} which is too small to be represented as a normalized number in so it is normalized to zero. but if the result is not normalized we will get the correct result.
- **Gradual underflow:** One of the advantages of the denormalized numbers is the gradual underflow. This came from the fact the normalized numbers can represent minimum numbers is 1.0×2^{min} and all numbers smaller than that are rounded to zero (which means there are no numbers between 1.0×2^{min} and 0 . The denormalized numbers expands the range and gives gradual underflow through the division of the range between 1.0×2^{min} to 0 with the same steps as the normalized numbers . For more information refer to [1] [2] [3]

6 Special values

The IEEE-754 standard supports some special values that gives special functions and give some signals.

Table of Special values

Name	Exponent	Fraction	sign	Exp Bits	Fract Bits
$+0$	$min - 1$	$= 0$	$+$	All zeros	All Zeros
-0	$min - 1$	$= 0$	$-$	All zeros	All Zeros
Number	$min \leq e \leq max$	any	any	Any	Any
$+\infty$	$max + 1$	$= 0$	$+$	All ones	All zeros
$-\infty$	$max + 1$	$= 0$	$-$	All ones	All zeros
NaN	$max + 1$	$\neq 0$	any	All ones	Any

6.1 Zero

- The zero is represented as a signed zero (-0 and $+0$)
- it is represented as $min - 1$ in the exponent and zero in the fraction.
- The signed zero is important for operations that preserves the sign like multiplication and division. It is also important to generate $+\infty$ or $-\infty$
- It is also used in the signum function that return the sign of a number.
- Event hough the standard defines the comparison $-0 = +0$ as true.

6.2 NaN

- Some computations generate undefined results like $0/0$ and $\sqrt{-1}$. These operations should be handled or we will get strange results and behavior. NaN is defined to be generated upon these operations and so the operations are defined for it to let the computations continue.
- Whenever a NaN participates in any operation the result is NaN.
- There is a family of NaN according to the above table and so the Implementations are free to put any information in the fraction part.
- All comparison operators $(=, <, \leq, >, \geq)$ (except $(\neq)$should return false when NaN is one of its operands.

Sources of NaN

Operation	Produced by
$+$	$\infty + (-\infty)$
$\times$	$0 \times \infty$
$/$	$0/0, \infty/\infty$
REM	$xREM0, \infty REMy$
$\surd$	$\sqrt{x}(whenx < 0)$

6.3 Infinity

- The infinity is like the NaN, it is a way to continue the computation when some operations are occurred.
- **Generation** Infinity is generated upon operations like $x/0$ where $x \neq 0$
- **Results:** The results of operations that get ∞ as parameter is defined as: "Replace the ∞ by the limit $lim_{x\to\infty}$. For example $3/\infty = 0$ because $lim_{x\to\infty}3/x = 0$ and $\sqrt{\infty} = \infty$ and $4 - \infty = -\infty$

- The infinity is used instead of the saturation on maximum representable number and the computation should continue.

- **Example[1]:** compute $\sqrt{x^2 + y^2}$ when max exponent is 98 and only three decimal digits are supported. If $x = 3 \times 10^{70}$ and $y = 4 \times 10^{70}$ and saturation is used $x^2 = 9.99 \times 10^{98}$ and so y^2. and so the final result it $(9.99 \times 10^{98})^{1/2} = 3.16 \times 10^{49}$ which is different than the correct result (5×10^{70}). Instead when Infinity is used, $x^2 = \infty$ and $y^2 = \infty$ so the final result is ∞ which is much better than giving incorrect result.

7 Exceptions

- Exceptions are important factors in the standard to signal the system about some operations and results.

- when an exception occurs, a status flag is set.

- The implementation should provide the users with a way to read and write the status flags.

- The Flags are **"sticky"** which means once a flag is set it remains until its explicitly cleared.

- The implementation should give the ability to install trap handlers that can be called upon exceptions.

- **Overflow, underflow and division by zero** are obvious from the table below. The distinction between Overflow and division by zero is to give the ability to distinguish between the source of the infinity in the result.

- **Invalid** This exception is generated upon operations that generates NaN results. But this is not a reversible relation (i.e. if the out put is NaN because one of the inputs is NaN this exception will not raise).

- **Inexact** It is raised when the result is not exact because the result can not be represented in the used precision and rounding cannot give the exact result.

- **Software flags** The inexact result exception is raised so often. So some implementations suggests that the hardware generates interrupts upon exceptions, while the software keeps the sticky status flags. In this case once an exception occurs, an interrupt is signaled to the software and the flag is set and that interrupt is masked. Once the flag is unset the interrupt is unmasked again.

Exceptions in IEEE 754 standard

Exception	Cased by	Result
Overflow	Operation produce large number	$\pm\infty$
Underflow	Operation produce small number	0
Divide by Zero	$x/0$	$\pm\infty$
Invalid	Undefined Operations	NaN
Inexact	Not exact results	Round(x)

8 Rounding modes

- The standard requires the following rounding modes: Round toward nearest, Round to zero, Round to $+\infty$, Round to $-\infty$.
- There are three types of round to nearest according to the standard
- Round to nearest even
- Round half-integers away from 0
- Truncate to integers towards 0

9 Comparison

- The standard requires the comparison to be exact (i.e. no overflow nor underflow)
- Four relations should be implemented: equal,less than, greater than, unordered
- the sign of zero should be ignored
- comparisons involving NaN should produce 'unordered'

References

[1] What every computer scientist should know about Floating-Point arithmetic. by David Glodberg. "http://www."

[2] Lectures notes on the Status of IEEE standard 754 for Binary Floating-Point Arithmetic. "http://www."

[3] An Interview with the Old Man of Floating-Point. "http://www.cs.berkeley.edu/ wkahan/ieee754status/754story.html"