# 

**64 BIT REPRESENTATION OF DECIMAL FRACTIONS IN MEMORY**

**BY**

**CHRISTABEL ANYANGO ONYANGO**

**IF/03/16**

**A RESEARCH PROJECT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR AWARD OF BACHELOR OF SCIENCE IN INFORMATICS**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**MOI UNIVERSITY MAIN CAMPUS**

**MARCH 2020**

# DECLARATION

**By Candidate**

This is to certify that the research is my original work and has not been presented for academic purposes in Moi University or any other institution of higher learning and has been submitted for examination.

**Name Reg.no Date Sign**

CHRISTABEL ANYANGO ONYANGO IF/03/16 ……… ……

**By Supervisor**

This research project has been submitted for examination with my approval as the supervisor

**Name Date Sign**

**DR. AUSTIN OWINO WANDERA** ……… …………

# ABSTRACT

The continuous advancement in computing and communication technologies over the 20th century has contributed to the current state of digital sophistication in this information age. By capitalizing on computer advances in microminiaturization, establishing and incorporating human computer interface key design principles in software development, the computing technology has become popular. It has spread to a wide range of users making it a fundamental business and basic need.

In a software application layout tailored to give a good user experience (UX), abstraction of the inner workings of the system components is usually one of the critical design rules adopted. This is to ensure simplicity and usability when interacting with the User Interface (UI). The UI usually hides the complexities of the underlying data structures and algorithms employed to perform given set of tasks. This may give most users only a casual familiarity of tasks performed by computer programs. e.g. When we enter a number in any software application, we normally don’t have to give any serious thought to how it will be represented in the computer. Most of the time we assume the representation in memory will be done correctly.

To uncover the black box beneath the abstraction, we set to find out the computations involved in representing decimal fractional numbers in a typical computer memory. Using guidelines provided by IEEE 754-2008 Standard, we specifically explore the double precision (64 bit, binary64) floating-point format. This format serves as the blueprint for the design and implementation of an algorithm that mimics the computations done by a typical computer to convert a fractional (floating-point) number of radix 10 (decimal notation) to its equivalent representation in memory of radix 2 (binary notation).

**TABLE OF CONTENTS**

[1](#_Toc33995779)

[DECLARATION 2](#_Toc33995780)

[ABSTRACT 3](#_Toc33995781)

[1.0: INTRODUCTION 5](#_Toc33995782)

[1.1: RESEARCH QUESTIONS 6](#_Toc33995783)

[1.2: AIMS AND OBJECTIVES 6](#_Toc33995784)

[1.3: SCOPE 7](#_Toc33995785)

[1.4: LIMITATION 7](#_Toc33995786)

[2.0: THIS WORK 8](#_Toc33995787)

[2.1: INTRODUCTION 8](#_Toc33995788)

[2.2: THE IEEE 754 STANDARD 9](#_Toc33995789)

[2.3: THE FLOATING-POINT REPRESENTATION 10](#_Toc33995790)

[2.4: 64-BIT (DOUBLE PRECISION, BINARY64) FLOATING-POINT FORMAT 11](#_Toc33995791)

[2.4.1: 64-BIT REPRESENTATION ALGORITHM 11](#_Toc33995792)

[2.4.2: THE 64-BIT REPRESENTATION PROGRAM 14](#_Toc33995793)

[3.0: CONCLUSION 15](#_Toc33995794)

[4.0: REFERENCE 16](#_Toc33995795)

[5.0: APPENDIX 17](#_Toc33995796)

[5.1: ACRONYMS AND ABBREVIATIONS 17](#_Toc33995797)

[5.2: STEP2: CONVERSION OF DECIMAL FRACTION NUMBER TO BINARY FRACTION. 17](#_Toc33995798)

[5.3: STEP 6: CALCULATION OF EXPONENT VALUE 19](#_Toc33995799)

# 1.0: INTRODUCTION

Numbers are the earliest way devised for keeping record of a count. There are different number systems that have been used over the centuries in different cultures to assign value and identify differences between items-especially in commerce. These numbers are however represented differently depending on the number of unique digits used to represent them (radix). In the course of history, the decimal system has overshadowed all others number systems to become the most popular positional number system used globally. It has established the basis for the development of not only the western commerce but the global market.

The decimal number system has a radix (base) of 10 with digits in the range 0 through 9. This number system represents integers and fractional values. Integers are counting numbers which provide the means to specify discrete quantities *e.g. 10* keyboards; whereas, fractions describe continuous quantities with a basic measurement unit intermediate between two whole numbers enabling representation of very small units *e. g* *2.31569456* GHz processor speed. These fractional values normally consist of a variant of integral numbers (integral part) and fractional numbers (fractional part). The decimal fractional (floating-point) numbers are being used frequently in commerce and everyday activities involving measurement; where amounts of a basic measurement unit depend on the range and precision required therein.

Following the advancement in Information and Communication Technology (ICT), the advent of computers has made the representation of decimal fractions be an indispensable tool for commerce and various organizational operations. We find that floating-point arithmetic is used extensively in many applications across multiple market segments. These applications often require a large number of calculations and are prevalent in finance and scientific research fields. Some of the fields include: bioinformatics, financial analytics, radar, molecular dynamics and space research*.*

In order to enhance portability and compatibility of floating-point representation across various hardware and software platforms, the IEEE-754 standard was established in 1985.This standard outlines the guidelines on implementations of both integers and fractions (floating-point numbers) across hardware manufacturers and software engineers.

## 1.1: RESEARCH QUESTIONS

* How are decimal fractional numbers stored/represented in a computer memory?
* What are the computations involved in the representation process in a 64-bit format?

## 1.2: AIMS AND OBJECTIVES

The main purpose of this project is to explore the methods employed to ensure correct approximation of representation of decimal fractional numbers in computer memory. Also, to design and implement an algorithm that mimics the representation process.

The objectives to be undertaken during the research project include:

* Identify and understand how decimal fractions are stored in memory.
* Research, understand and describe the computations that may be involved during the representation process.
* Gain an understanding of relevant algorithmic concepts defined in IEEE-754 Standard for double precision (binary64) format.
* Design and implement an algorithm that takes a decimal fractional number as input and returns its equivalent binary notation in memory using the binary64 format representation.
* Display transformations of the input value until the final value(output) is returned.

## 1.3: SCOPE

In this research project, we specifically look into decimal fractional (floating point) numbers representation on the computer memory. We also explicitly use guidelines provided by the IEEE 754-2008 standard for the double precision(binary64) format.

## 1.4: LIMITATION

* **Insufficiency in funds and key resources.**

Most professional algorithmic simulation software applications require premium access. With an underestimation of cost, type and availability of resources to enable simulation, we could be constrained to use a web application for visualization of the algorithm.

* **Insufficient time**

The technicality of the research subject requires time to get a firm grip of concepts. The consultations and development take a considerable time as well. This can be arduous and time demanding especially when undertaking the project concurrently with other semester’s courses.

# 2.0: THIS WORK

## 2.1: INTRODUCTION

Decimal fractions are normally used where precision and range are critical to preserve integrity in quality and quantity of items. These decimal fractions could either be terminating decimal fractions or non-terminating decimal fractions. The fractional parts of these numbers could stretch to as far as the required precision of our imaginative calculations (infinite brain memory). The precision used to control the size of the fractional part differs among various disciplines with strains and conflicts between range, precision and the complexity of numbers in arithmetic the human brain can manage at a given time.

In the advent of computers, management of complexity in calculation was minimized, however the memory and other computing resources used are finite. Computers can only hold a finite amount of information. It has storage capacities ranging from bytes to yottabytes [*1 B =1,208,925,819,614,629,174,706,176 YB]*. Also, the computers smallest unit of information has two states. The states are derived from various processes of its components e.g. magnetic storage devices have areas that are either magnetized or not .These two states can be represented by the binary digits (bits) as 1 and 0. Bits being the building blocks for all information processing in computers, the binary number system became the de facto machine language .Data and information are therefore represented in binary notation in a computer system.

Moreover, computers are often classified by the number of bits they can process at one time as well as by the number of bits used to represent addresses in their main memory (RAM). The number of bits used by a computer’s CPU for addressing information, represents one measure of a computer’s speed and power.As opposed to single-precision32-bit floating-point representation, many applications demand higher precision, forcing the use of double-precision64-bit operations. For computers that use the 64 bits in their addressing; a 64-bit processor, for example, has 64-bit registers, 64-bit data buses, and 64-bit address buses. This means that a 64-bit address bus can potentially access up to = 18,446,744,073,709,551,616 memory cells (bytes) in RAM. With this finite units of storage, we can only have an approximation of decimal fractional numbers represented in memory. The more bits we use, the better the approximation. Therefore, when representing floating-point values, we model the infinite system of real numbers into a finite system of integers. To guide this implementation across various hardware and software platforms, the IEEE 754 floating point representation standard was established.

## 2.2: THE IEEE 754 STANDARD

The IEEE Standard 754, adopted in 1985 is a technical guideline for implementations of floating-point arithmetic. Since its establishment, this standard has been revised in subsequent years 2008 and 2019 by the Institute of Electrical and Electronics Engineers (IEEE). The first version of the standard covered only binary floating-point arithmetic. The IEEE 754- 2008 includes nearly all of the original IEEE 754- 1985 standard guidelines and the IEEE 854-1987 Standard for Radix-Independent Floating-point arithmetic. The current version (IEEE-754 2019) contains minor revision of the previous version, incorporating mainly clarifications, defect fixes and new recommended operations.

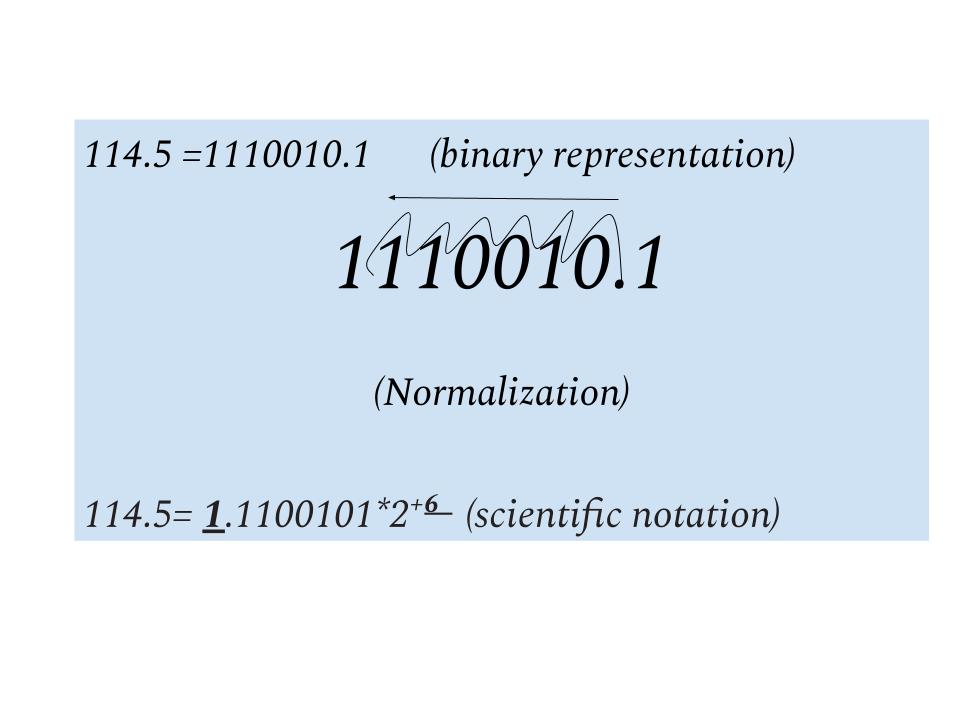
The standard facilitates the portability of programs from one processor to another and also encourages the development of sophisticated, numerically oriented programs.

Previously before the advent of the standard, decisions on acceptable range and precision of floating- point values were purely random. This resulted in numerous incompatible representations across various manufacturers’ systems. For instance, for well-established architectures such as IBM mainframes, IBM systems had been using the same architecture for floating-point arithmetic that the original System/360 used in 1964 which employed 36-bit format. They later made the decision to move to a 32-bit architecture owing to portability and wide adoption of the IEEE 754 standard on various contemporary processors and arithmetic coprocessors manufacturers.

Despite the various versions of the IEEE 754 Standard, we base this project research on IEEE 754 -2008. This standard defines the following different types of floating-point formats:  
arithmetic format, basic format, interchange format and extended format. Since the project research is based on 64 -bit representation, we explore the basic format. This format covers five floating-point representations, three binary and two decimals. The three binary formats have bit lengths of 32, 64, and 128 bits, with exponents of 8, 11, and 15 bits, respectively. Of the three formats, the 64-bit format is the basis to find the underlying solutions to our research questions.

## 2.3: THE FLOATING-POINT REPRESENTATION

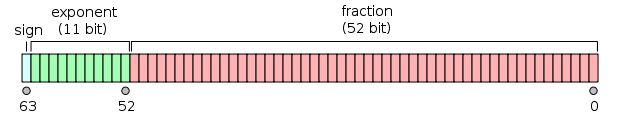
The floating-point representation is a type of representation that employs normalization of binary fractions. Normalization involves shifting of the binary point to the right of the most significant bit and having the exponent keep track of the shifts as it adjusts its value by 1 for every shift.



This type of representation is stipulated in the IEEE 754 2008 standard and is used to represent the three binary formats (32,64 and 128).

## 2.4: 64-BIT (DOUBLE PRECISION, BINARY64) FLOATING-POINT FORMAT

The double precision format is 64 bits in length and contains 3 fields critical in managing and maintaining the precision and range of numbers during various conversions and storage. The fields include: sign (**+** or **-**), exponent (**E**) and significand (**S**)

****

***Figure 1*: *Double Precision Format (Source: Nanyang Technological University (Singapore), 2014)***

We use the guidelines provided by the IEEE 754-2008 double precision format(64-bit) that form the basis for the design and implementation of the algorithm.

### 2.4.1: 64-BIT REPRESENTATION ALGORITHM

Decimal fractional value= {Input N}

**Conversion of Decimal fraction to binary fraction**

1. **{Separate integral and fractional parts of N}.** We Separate the integral part and the fractional part of the decimal fractional value.
2. **{Convert int\_N and float\_N to binary}.** We Convert the Integral and Fractional parts of the decimal fraction to binary notation.

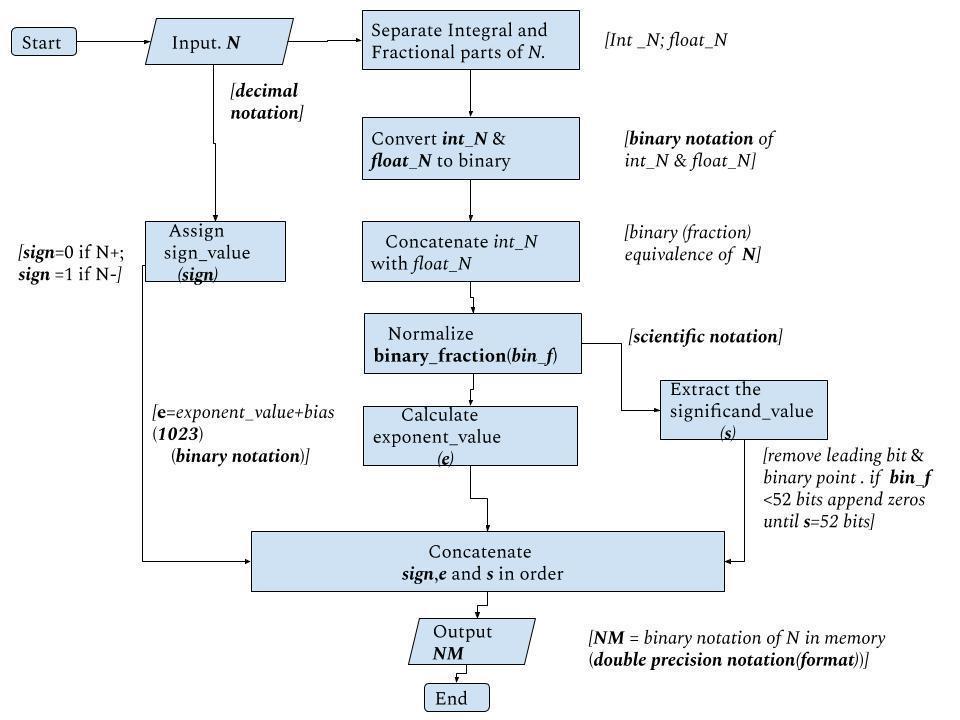
*See step2 from the Appendix [illustration]*

1. **{Concatenate int\_N and float\_N}.** We Concatenate the output of both the integral and fractional parts of **STEP2** and insert a binary point (the radix point) between them. If the resulting number of bits is greater than 53 bits; Right trim the output of concatenation

**Normalization of Binary Fraction**

1. **{Normalize the binary\_fraction}.** We Shift the radix point by 1 bit to the right or left of its original position in **STEP3’s** output. Repeat this until 1 is the only digit to the left of the radix point.

[ *The resulting fractional number is normal. The total number of shift(s) is the* ***exponent***. *Shifts to the right result in negative exponent while shifts to the left result in positive exponent*]

****

***Figure 2*: *64-bit memory representation algorithm design***

**Assign values to IEEE Double Precision representation fields**

1. **Sign Bit**
2. **{Assign *sign* value}.** We set the ***sign*** to **1** otherwise we set it to **0**

// This step can be done in parallel with **STEP1 //**

[ *Positive fraction values=0; Negative fraction values=1.The field is* ***1 bit*** *in length*]

1. **Biased Exponent**
2. **{Calculate exponent value}.** We Add the exponent from **STEP4** (true exponent) to a fixed value bias of **1023** to get the biased representation of the exponent (biased exponent) in decimal notation.

[*The exponent is stored in the range 1 -2046 (0 and 2047 have special meanings). The true exponent values are in the range -1022 to + 1024. The bias is used to ensure that all stored exponents are positive in the final representations and are treated as unsigned integers.*]

We Convert the output of **STEP6** to binary notation using the procedure in **STEP2** (integral conversion). *See STEP 2 figure 3 from the Appendix [illustration].* This is the output of this step.

[ *The output is the exponent representation in memory consisting of* ***11 bits.***]

*See STEP 6 figure 4 from the Appendix [illustration]*

2. **Significand Representation**
3. **{Extract the significand value}.** We Trim the normalized binary fraction (**STEP4** output) to remove the leading 1 and radix point and record the rest of the significant digits. If need be (the rest of the significant digits are less than 52 bits), append zeros to the significant digits until the bits are 52. The output is the significand representation.

[*The leading 1 is implicit and is not stored on the significand field (the hidden bit convention) because it is always of the same value i.e. 1 is always the one bit to the left of the radix point, for all normalized binary fractional numbers.* *Therefore, the 52-bit field is used to store a 53-bit significand with the value the leading 1 not included in the field]*

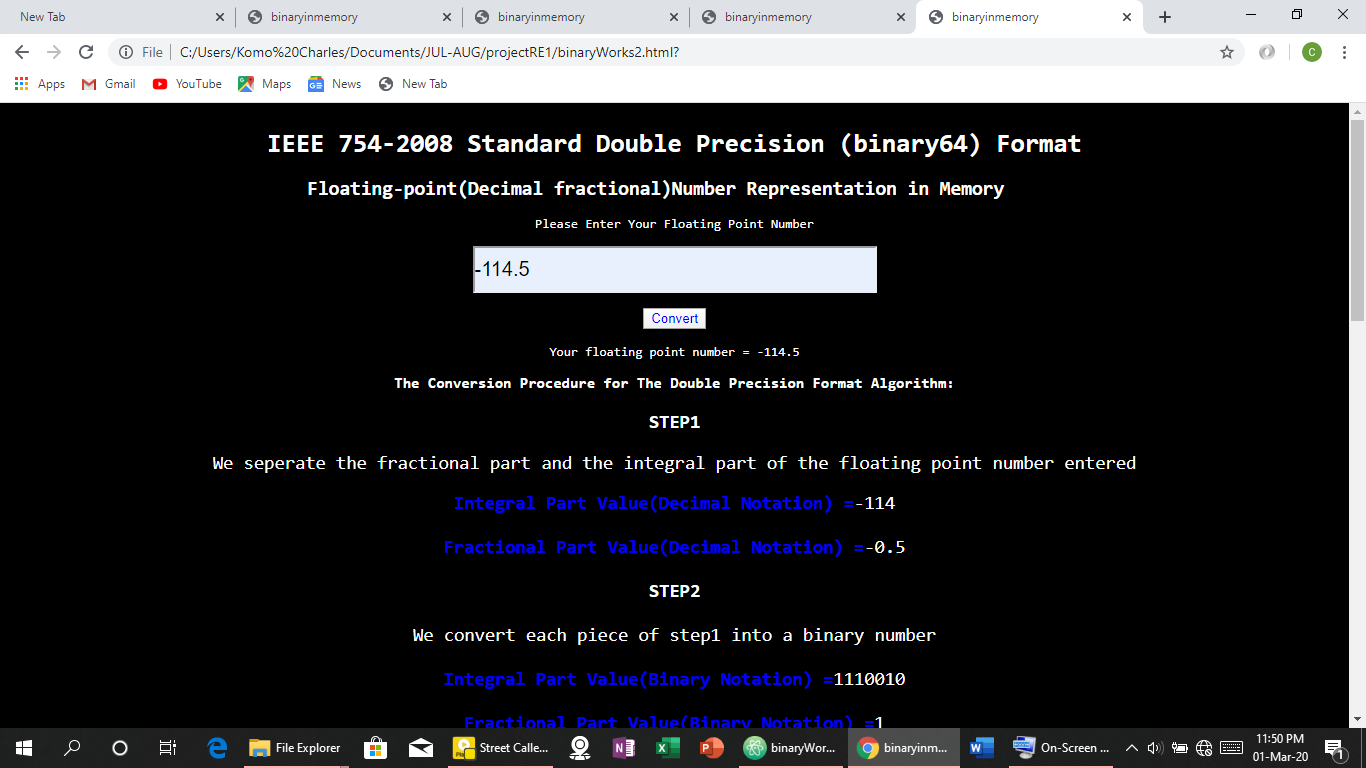
**Pattern Representation in Memory**

1. **{Concatenate sign, exponent and significand}.** We Concatenate the output of **STEPS 5 6** and **7** in that sequential order. This is the double precision format representation of decimal fractions in memory.

[*The resultant binary string is the representation of floating-point numbers in memory for the double precision format (64 bit)*]

### 2.4.2: THE 64-BIT REPRESENTATION PROGRAM

The program developed from the algorithm above, accepts a decimal fractional value as its input value. The program then displays the progressive transformation of the input value at each step of the procedure. The final output value displayed is the 64-bit representation of the decimal fractional number in memory.



***Program snapshot showing input value transformations at step1 and 2***

# 3.0: CONCLUSION

Representing a fractional number of decimal notation in memory is not as obvious as just converting a number to its equivalent binary value. The algorithms employed to enable floating-point representations on our computer systems are developed using the IEEE 754 Standard guidelines. Despite being constrained by the computer’s finite memory available for representation of decimal fractional numbers, the IEEE 754 standard provides a variety of acceptable representation formats. These formats have a variance in range and precision of decimal fractional numbers representations they can attain upon implementation. We therefore conclude that the type of format used on given computing devices (floating-point units) such as PCs or supercomputers depend on the level of accuracy in floating point representation required by users of the system(s).

# 4.0: REFERENCE

1. Nanyang Technological University. (2014). *The Bit String Encoding in Memory for Double Precision Format.* [Drawing] Retrieved from <https://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html>
2. Deepayan Sarkar.( 2016).*The Floating Point Representation for Double Precision Format in Memory*.[Drawing] Retrieved from <https://www.isid.ac.in/~deepayan/ICP2016/notes.html>
3. Null, L. & Lobur, J. (2003). Data Representation in Computer Systems. In *The Essentials of Computer Organization & Architecture.* (pp. 55-61). Sudbury, MA: Jones and Bartlett.
4. Stallings. (2016). Computer Arithmetic. In P, Zeno (Ed).  *Computer Organization & Architecture: Designing for Performance* (10th ed., pp 350-357). Hoboken, NJ: Pearson Education, Inc.
5. Kreindler, D (2007). *How to implement double-precision floating-point on FPGA*. Retrieved from <https://www.eetimes.com/how-to-implement-double-precision-floating-point-on-fpgas/>
6. Michael L. Overton. (1996). Computer Representation of Numbers. In *Floating Point Representation.*

# 5.0: APPENDIX

## 5.1: ACRONYMS AND ABBREVIATIONS

IEEE Institute of Electrical and Electronics Engineers

UI User Interface

CPU Central Processing Unit

YT Yottabytes

B Bytes

UX User Experience

PC Personal Computer

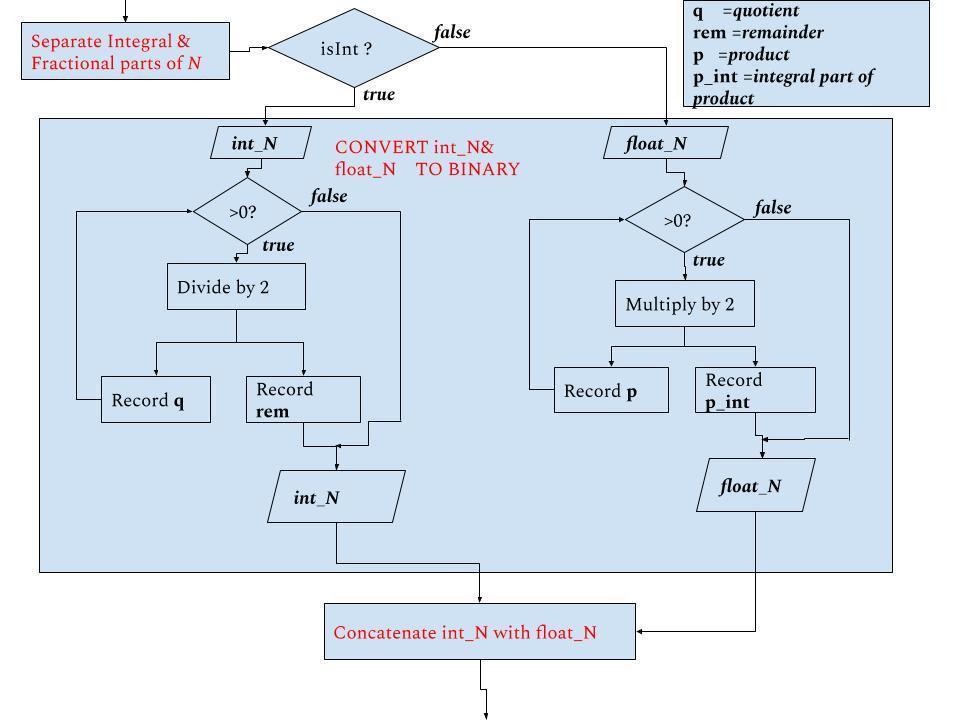
## 5.2: STEP2: CONVERSION OF DECIMAL FRACTION NUMBER TO BINARY FRACTION.

**INTEGRAL PART OF NUMBER *(Input N)***

1. Divide the integral value by 2 and record the resultant quotient and the remainder (in a sequence).
2. Repeat this with each resulting quotient as the dividend and halt when the quotient value is zero.
3. Reverse the remainders to form the binary notation of the integral part of the floating-point number

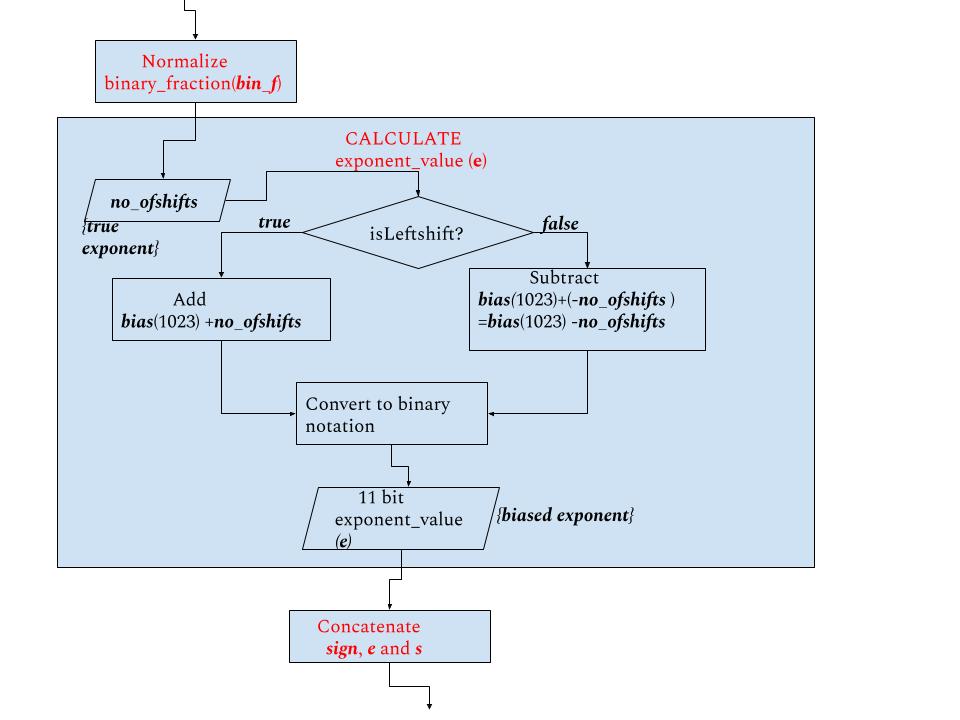
**FRACTIONAL PART OF NUMBER *(Input N)***

1. Multiply the fractional part value by 2 and record the product and the product’s integral part value (in a sequence).
2. Repeat this with each resulting product’s fractional part value as the multiplicand and halt when the product value is zero (if terminating binary fraction) or when the recorded products integral part value bits are 53(repeating binary fraction).



***Figure 3*: *Conversion of the integral part and fractional part of a decimal number to binary notation***

## 5.3: STEP 6: CALCULATION OF EXPONENT VALUE



***Figure 4: Calculation of the biased exponent representation (e)***