Computer Arithmetic in modern computers and usages of Computer Arithmetic

Pouya Shams Ahari
Department of Electrical Engineering. Ahar Branch, Islamic Azad University, Ahar, Iran
pouyashams1@yahoo.com

1. Introduction

As the ability to perform computation increased from the early days of computers and up to the present so was the knowledge how to utilize the hardware and software to perform computation. [2]Digital computer arithmetic emerged from that period in two ways: one as an aspect of logic design and other as development of efficient algorithms to utilize the available hardware.

Given that numbers in a digital computer are represented as a string of zeroes and ones and that hardware can perform only relatively simple and primitive set of Boolean operations, all of the arithmetic operations performed are based on a hierarchy of operations that are built upon the very simple ones.

What distinguishes computer arithmetic is its intrinsic relation to technology and the ways things are designed and implemented in a digital computer. This comes from the fact that the value of a particular way to compute, or a particular algorithm, is directly evaluated from the actual speed with which this computation is performed. [4] Therefore there is a very direct and strong relationship between the technology in which digital logic is implemented to compute and the way the computation is structured. This relationship is one of the guiding principles in development of the computer arithmetic.

2. Computer Arithmetic

Arithmetic is a branch of mathematics that deals with numbers and numerical computation. Arithmetic operations on pairs of numbers x and y include :
addition, producing the sum $s = x + y$,
subtraction, yielding the difference $d = x – y$,
multiplication, resulting in the product $p = x \times y$,
and division, generating the quotient $q = x / y$.
Subtraction and division can be viewed as operations that undo the effects of addition and multiplication, respectively.
Computer arithmetic is a branch of computer engineering that deals with methods of representing integers and real values (e.g., fixed- and floating-point numbers) in digital systems and efficient algorithms for manipulating such numbers by means of hardware circuits or software routines. [1] On the hardware side, various types of adders, subtractors, multipliers, dividers, square-rooters, and circuit techniques for function evaluation are considered. Both abstract structures and technology-specific designs are dealt with. Software aspects of computer arithmetic include complexity, error

characteristics, stability, and certifiability of computational algorithms.

Of the various types of data that one normally encounters, the ones we are concerned with in the context of scientific computing are the numerical types.

Computer hardware is organized to give only a certain amount of space to represent each number, in multiples of bytes , each containing 8~ bits . Typical values are 4 bytes for an integer, 4~or~8 bytes for a real number, and 8~or~16 bytes for a complex number.

Since only a certain amount of memory is available to store a number, it is clear that not all numbers of a certain type can be stored. For instance, for integers only a range is stored. (Languages such as Python have arbitrarily large integers , but this has no hardware support.) In the case of real numbers, even storing a range is not possible since any interval [a,b] contains infinitely many numbers. Therefore, any representation of real numbers will cause gaps between the numbers that are stored. Calculations in a computer are sometimes described as finite precision arithmetic . [1] Since many results are not representable, any computation that results in such a number will have to be dealt with by issuing an error or by approximating the result. In this chapter we will look at the ramifications of such approximations of the `true' outcome of numerical calculations.

3. History of Computer Arithmetic

Gottfried Leibniz

• Discovered the binary system in 1679

George Boole

• Wrote a paper about

Boolean algebra in 1854

• Boolean algebra would become important later for the digital computer

Claude Shannon

• Created a Boolean algebra and binary arithmetic circuit in 1937

• First practical application to digital circuit design

George Stibitz

• Worked for Bell Labs

• Created a relay based computer called “Model K” in 1937

(k stood for kitchen)

Numbers less than zero?

Several ways to represent negative numbers

•Sign-and-Magnitude

•One’s Complement

•Two’s Complement

•Excess-K

•Base -2

Numbers less than zero! [3]

• Sign-and-Magnitude was popular for a while

o Intel 7090 Architecture (1959) used it

• 2’s complement won out in the end

o Easier to implement in hardware

o Unique representation of zero
Architectures using 2’s complement

• x86, MIPS, ARM...

Addition

Need for faster ways to add

• Ripple Carry (slow)

• Carry-Lookahead

oGerald Rosenberger (1957)

oExamples

Manchester Carry Chain

Brent-Kung Adder

Kogge-Stone Adder

Multiplication

•Until the 1970s, computers did not have multiplication hardware

•Motorola 6809 was one of the first processors to have a multiply instruction (routine in microcode)

•As transistor count increased, it became feasible to include enough adders to sum all partial product simultaneously.

Division Algorithms[3]

• Two types

o Fast

o Slow

• Slow division

oRestoring division

oNon-restoring division

oSRT division

• Fast division

o Newton-Raphson division

o Goldschmidt division

We need more precision!

• Integers just aren’t good enough!

• Floating Vectors

oJanes H. Wilkinson (1951)

oScaled to biggest number

oProblem when dealing with a big range of numbers

• Fixed point

• Floating Point

oPrevalent by 1957

oDecimal floating-point

oBinary floating-point

Floating Point

•By 1976, floating point functions were everywhere, but very proprietary

•IEEE began trying to standardize floating point in 1977

•Because there were business majors working in IEEE, it wasn't until 1985 that IEEE Standard 754 was put forth.

4. usages of Computer Arithmetic

Data is manipulated by using the arithmetic instructions in digital computers. Data is manipulated to produce results necessary to give solution for the computation problems. The Addition, subtraction, multiplication and division are the four basic arithmetic operations. If we want then we can derive other operations by using these four operations.

To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed generally on binary or decimal data. Fixed-point numbers are used to represent integers or fractions. [4] We can have signed or unsigned negative numbers. Fixed-point addition is the simplest arithmetic operation.

If we want to solve a problem then we use a sequence of well-defined steps. These steps are collectively called algorithm. To solve various problems we give algorithms.

In order to solve the computational problems, arithmetic instructions are used in digital computers that manipulate data. These instructions perform arithmetic calculations.

And these instructions perform a great activity in processing data in a digital computer. As we already stated that with the four basic arithmetic operations addition, subtraction, multiplication and division, it is possible to derive other arithmetic operations and solve scientific problems by means of numerical analysis methods.

A processor has an arithmetic processor(as a sub part of it) that executes arithmetic operations. The data type, assumed to reside in processor, registers during the execution of an arithmetic instruction. Negative numbers may be in a signed magnitude or signed complement representation. There are three ways of representing negative fixed point - binary numbers signed magnitude, signed 1's complement or signed 2's complement. Most computers use the signed magnitude representation for the mantissa .

Until now, we have worked with data as either numbers or strings. Ultimately, however, computers represent everything in terms of binary digits, or bits. A decimal digit can take on any of 10 values: zero through nine. [3] A binary digit can take on any of two values, zero or one. Using binary, computers (and computer software) can represent and manipulate numerical and character data. In general, the more bits you can use to represent a particular thing, the greater the range of possible values it can take on.

Modern computers support at least two, and often more, ways to do arithmetic. Each kind of arithmetic uses a different representation (organization of the bits) for the numbers. The kinds of arithmetic that interest us are:

5. Decimal arithmetic

This is the kind of arithmetic you learned in elementary school, using paper and pencil (and/or a calculator). In theory, numbers can have an arbitrary number of digits on either side (or both sides) of the decimal point, and the results of a computation are always exact.

Some modern systems can do decimal arithmetic in hardware, but usually you need a special software library to provide access to these instructions. There are also libraries that do decimal arithmetic entirely in software.

6. Integer arithmetic

In school, integer values were referred to as "whole" numbers—that is, numbers without any fractional part, such as 1, 42, or -17. The advantage to integer numbers is that they represent values exactly. The disadvantage is that their range is limited.

In computers, integer values come in two flavors: signed and unsigned. Signed values may be negative or positive, whereas unsigned values are always greater than or equal to zero.

In computer systems, integer arithmetic is exact, but the possible range of values is limited. [3] Integer arithmetic is generally faster than floating-point arithmetic.

7. Floating-point arithmetic

Floating-point numbers represent what were called in school "real" numbers (i.e., those that have a fractional part, such as 3.1415927). The advantage to floating-point numbers is that they can represent a much larger range of values than can integers. The disadvantage is that there are numbers that they cannot represent exactly. [2]

Modern systems support floating-point arithmetic in hardware, with a limited range of values. There are software libraries that allow the use of arbitrary-precision floating-point calculations.

POSIX (It is a set of standards defined for naming and defining an application programming icon in Unix-like environments in IT triplets) uses double-precision floating-point numbers, which can hold more digits than single-precision floating-point numbers.

Computers work with integer and floating-point values of different ranges. Integer values are usually either 32 or 64 bits in size. Single-precision floating-point values occupy 32 bits, whereas double-precision floating-point values occupy 64 bits. (Quadruple-precision floating point values also exist. They occupy 128 bits.) Floating-point values are always signed.

Conclusions

We may divide mathematics into three categories—natural mathematics, computational or, equivalently, computer mathematics, and mathematics. Natural mathematics is the mathematics used by nature all the time throughout the universe. This mathematics follows all the laws of nature (known or unknown to us), knows no error, represents all quantities exactly, almost all of which can never be captured by any

means by any human being/living being, and uses infinite precision always for all computations. Computer mathematics, on the other hand, follows only known man-made rules, knows only errors, represents almost all quantities nonexactly, and uses finite precision always for all computations. [4]Mathematics used by a mathematician are often expressed symbolically, tends/attempts to capture some of the aspects of natural mathematics, and uses the much fewer laws of nature known to human beings. While it can never capture error exactly, it symbolically provides bounds for an error; such bounds may or may not be useful in practice.

The computer arithmetic—a vital feature of computer mathematics—is essentially the IEEE 754 floating-point arithmetic. It often uses additional features, taking full advantage of binary representation in the hardware computer. The IEEE standard specifies three formats—single (32 bits), double (64 bits), and double-extended—of floating-point numbers. Each format can represent +0, −0, NaN (Not-a-Number), ±∞ (infinity), and its own set of finite real numbers all of the simple form 2k+1−Nn with two integers n (signed significant) and k (unbiased signed exponent) that run through two intervals determined from the format. Each of zero and infinity has two representations besides NaN (produced when, for instance, division by zero is encountered) in this representation.

Besides IEEE 754 which is a binary standard, there is the IEEE 854 .That represents the conventional number system used and thoroughly understood by humans all over the globe and is specially suitable for calculators. Unlike IEEE 754, it does not specify how floating-point numbers are encoded into bits. The 854 standard specifies constraints on allowable values of the finite precision p for the single precision as well as for the double precision but it does not need a particular value for p.

The role of universal (absolute/unique) zero and that of numerical nonabsolute (nonunique) zero are well adapted in both IEEE 754 and IEEE 854 standards in a computer. This adaptation ensures a best computational accuracy (least computational error) for a specified precision. It can be seen that zeros—both absolute and numerical—play a vital role not only in improving accuracy and detecting illegal arithmetic operations (such as NaN's) but also in taking care of an overflow and/or an underflow appropriately. Consequently the possibility of undetected computational mistakes entering into the sequence of computations resulting in wrong/unacceptable outputs/results is eliminated/minimized unlike many pre-IEEE standards.

REFERENCES

[1] http://www.wmgallery.com/stibitz/
[2] http://www.loria.fr/zimmerma/mca
[3] http://www.kerryr.net/pioneers/leibniz.html
[4] http://en.wikipedia.org/wiki/Binary_number #History