On the Fundamental Acts of Computation and Cognition: Addition and Comparison

Jose Portillo

Abstract

We explore the reduction of complex mathematical operations, digital computations, and cognitive processes to two fundamental acts: *addition* and *comparison*. This minimalist perspective unifies human arithmetic intuition, machine computation, and artificial intelligence under a simple conceptual framework.
This article is the product of a conversation between the author and an AI (GPT) subject: the human shares ideas and reflections, then asks the AI for its perspective. The final presentation you'll read here is synthesized by the AI based on that collaborative dialogue.

1 Introduction

Mathematics, computation, and cognition often appear as intricate domains requiring diverse operations and sophisticated algorithms. Yet, beneath this complexity lies a surprisingly simple core: the operations of *addition* and *comparison*.
In this article, we argue that all human or machine arithmetic operations, as well as fundamental AI processes, can be decomposed into these two basic acts.

2 Addition: The Universal Builder

Addition is the cornerstone of arithmetic, both in machine logic and human reasoning.

In Machines: Addition Beneath All

At the hardware level, all digital arithmetic operations are ultimately built upon the *adder* circuit. Consider the following:

- **Subtraction** is implemented as addition of the complement (e.g., two's complement).
- **Multiplication** is repeated addition.
- **Division** is repeated subtraction, which reduces to repeated addition via complements.
- **Exponentiation** is repeated multiplication, hence a tower of additions.

Thus, addition forms the *fundamental operation* underlying all numeric manipulation at the digital level.

In the Human Mind: Counting Forward

Surprisingly, the human mind does not stray far from this principle. Subtraction, for instance, is rarely performed directly — it is experienced as an act of counting upward from one number to another.
Take the example $5 - 3$. Most people, especially when first learning arithmetic, do not "recall" the difference. Instead, they think:

> *"Which number is smaller? And how many steps does it take to reach the other?"*

This is a cognitive act of addition: counting forward from 3 to 5.
Even multi-digit subtraction relies on similar transformations. When subtracting 586 from 940, we might first decompose the minuend:

$$940 = 900 + 40 + 0 = 800 + 30 + 10$$

This decomposition (or "borrowing") is a restructuring to allow subtraction — but its core remains an additive transformation. We are simply rebalancing the digits so that each component can be "subtracted," which again is operationally nothing more than adding up to the larger number.
Moreover, just like machines, humans understand there is no such thing as subtracting from zero. You can't take something away from nothing unless you borrow — or reinterpret the question. We reframe the subtraction as an additive gap-filling act, not as a direct removal.

To subtract is to count — to add toward a comparison

Thus, in both silicon and synapse, addition is not just a tool — it is the **default act**. Whether encoded as electrical pulses in hardware or mental steps in the brain, addition is the universal builder of all mathematical understanding.

3 When the Subtrahend is Greater: Human Arithmetic and Cognitive Borrowing

Let us now consider a seemingly simple question: what happens when we try to subtract a bigger number from a smaller one, say $5 - 8$?

In pure mathematical terms, the answer is negative: -3. But for a learning human, especially a child or anyone working with natural numbers, this case creates a conceptual hurdle. One does not "subtract 8 from 5" naturally — instead, we flip the question:

"How much is 8 more than 5?"

And then we tag the result as "negative." Once again, this reflects a comparison followed by a counting act. In essence:

$$5 - 8 = -(8 - 5)$$

So, when humans confront "subtraction in the wrong order," we instinctively perform a *comparison*, followed by an *addition* — and finally, adjust the sign.

This mirrors exactly what computers do via complement systems like two's complement. Negative numbers aren't handled by "going backward" — rather, they're transformed into a format that allows the adder to do its job, as if everything were positive, and then interpreted appropriately.

Thus, subtraction—whether it's $9 - 2$ or $2 - 9$—is never about direct removal. It's about **adding up to** or **counting the gap**, then optionally flipping direction.

And just like that, subtraction reveals itself once more to be a synthetic construction of our two fundamental operations:

$$\boxed{\text{Subtraction} = \text{Comparison} + \text{Addition}}$$

4 Comparison: The Gatekeeper of Logic

While addition combines quantities, *comparison* decides the flow of computation:

- Comparisons determine control structures in algorithms.
- Neural networks, despite their complexity, rely on comparisons (thresholding, argmax) to make decisions.
- Hardware logic units depend on comparison circuits to control data flow.

Therefore, comparison acts as the *fundamental decision-making* operation.

5 Synthesis: Addition and Comparison as the Essence

From human cognition, where subtraction is often performed by mentally counting the difference, to computer processors that use addition and comparison gates, the vast landscape of computation reduces to these two primal acts.

$$\boxed{\text{Computation} \equiv \underbrace{\text{Addition}}_{\text{Combine}} + \underbrace{\text{Comparison}}_{\text{Decide}}}$$

This duality serves as a guiding principle for understanding and teaching arithmetic, programming, and artificial intelligence.

6 IEEE-754: No Subtraction, Only Addition in Disguise

At the heart of every Intel, AMD, or NVIDIA processor lies the IEEE-754 standard, governing floating-point arithmetic. Despite its sophistication, IEEE-754 does not introduce fundamentally new arithmetic operations. Instead, it reinforces our thesis: all computation reduces to addition and comparison.

Consider subtraction. In the physical world, we understand that one cannot subtract from nothing — if you have zero apples, you cannot give any away. Digital hardware reflects this truth. There is no such thing as "subtracting from zero." Instead, subtraction is implemented by adding the *complement* of a number. In two's complement or IEEE-754 floating-point format, a negative number is simply another encoding of a positive number — one that, when added, has the effect of subtraction.

$$a - b \equiv a + (\sim b + 1)$$

Thus, subtraction is not an independent operation. It is addition under disguise — dressed up in a complement suit.

Moreover, in floating-point arithmetic, every subtraction begins with a comparison: the exponents must be aligned, the significands adjusted, and signs interpreted. Each of these steps relies on a cascade of comparison and addition operations. The complexity is real, but the building blocks remain the same.

Even multiplication and division—seemingly more complex operations—ultimately submit to the same principles:

- **Multiplication** of floating-point numbers is carried out by multiplying significands (via repeated or optimized addition) and adding exponents. At the lowest level, techniques such as Booth's algorithm or shift-and-add methods demonstrate that multiplication is just structured addition.
- **Division** works by repeated subtraction, often implemented through addition of complements. Advanced algorithms like Newton-Raphson or Goldschmidt iterations use iterative approximations, each step relying on addition and comparison to converge toward the quotient.
- **Exponentiation**, often implemented through repeated multiplication, reduces to layers of repeated addition.

IEEE-754 is often seen as a pinnacle of numerical engineering. And yet, it too submits to the minimal law of computational gravity:

$$\boxed{\text{All Arithmetic} = \text{Addition} + \text{Comparison}}$$

Even the giants of floating-point computation stand on the shoulders of these two basic acts.

7 Conclusion

By recognizing addition and comparison as the fundamental building blocks, we achieve a unifying perspective on the nature of computation and cognition. This insight encourages a minimalist yet powerful approach to both learning and designing computational systems.

Appendix A. Integer vs Floating-Point Arithmetic in Real Hardware

In actual modern computer hardware, like processors made by Intel, AMD, or NVIDIA, arithmetic is performed using dedicated hardware circuits designed for either integers or floating-point numbers. These are separate units on the chip, each optimized for their respective data types.

Integer Arithmetic

Integer operations (such as addition, subtraction, multiplication, and logical bitwise operations) are handled by the **Arithmetic Logic Unit (ALU)**. The ALU operates using binary arithmetic based on the two's complement representation. Integer arithmetic does not use IEEE-754; instead, it relies on fixed-width binary representations, usually in two's complement format.
Key features of integer units:

- Use fixed-size integers (8, 16, 32, or 64 bits).
- Perform operations like `ADD`, `SUB`, `AND`, `OR`, `XOR`, `SHL`, `SHR`.
- Efficient and fast, especially for control logic, memory addresses, and loop counters.

Floating-Point Arithmetic

Floating-point operations use a completely different circuit called the **Floating-Point Unit (FPU)**. These units are compliant with the IEEE-754 standard and handle numbers represented with sign, exponent, and significand.
Key features of FPUs:

- Handle real numbers (e.g., `float`, `double`).
- Operate with IEEE-754 compliant logic for rounding, normalization, infinities, and NaNs.
- Support operations like `FADD`, `FSUB`, `FMUL`, `FDIV`, `FSQRT`, etc.

Separation of Responsibilities

Because floating-point arithmetic is more complex (due to exponent alignment, rounding modes, and normalization), it would be inefficient to mix both types of arithmetic in a single hardware unit. Instead, processors maintain distinct execution pipelines for integer and floating-point instructions. This allows parallelism and greater performance.
Note: Some GPUs also have mixed-precision units to optimize performance on machine learning workloads, supporting formats like `bfloat16` or `tf32`, but they are still based on IEEE-754 principles.

IEEE-754 Floating-Point Arithmetic

For real numbers represented in IEEE-754 format, subtraction and addition are performed through similar procedures, primarily using a single addition circuit. Subtraction is handled by negating the sign bit of the second operand and then performing addition.
Each floating-point number is represented as:

$$x = (-1)^s \cdot 1.f \cdot 2^e$$

where:

- s is the sign bit
- f is the fraction (also called the significand)
- e is the biased exponent

Steps for IEEE-754 Addition/Subtraction (e.g., $A \pm B$)

1. **Flip the sign bit** of B if performing subtraction.
2. **Align the exponents** by shifting the significand of the operand with the smaller exponent.
3. **Add or subtract the significands**, depending on sign bits.
4. **Normalize the result** (adjust exponent and shift significand if needed).
5. **Round the result** to the nearest representable value.
6. **Repack the final result** into IEEE-754 format.

Example 1: Addition

Add:

$$A = 1.0 \times 2^2 = 4.0, \quad B = 1.0 \times 2^1 = 2.0$$

IEEE-754 (simplified):

$$A = 1.0 \times 2^2 \quad \text{(Exponent = 2)}$$
$$B = 1.0 \times 2^1 \quad \text{(Exponent = 1)}$$

Align exponents:

$$B = 0.5 \times 2^2$$

Now add:

$$A + B = (1.0 + 0.5) \times 2^2 = 1.5 \times 2^2 = 6.0$$

Result: 6.0

Example 2: Subtraction

Subtract:

$$A = 1.0 \times 2^3 = 8.0, \quad B = 1.0 \times 2^2 = 4.0$$

IEEE-754 (simplified):

$$A = 1.0 \times 2^3 \quad \text{(Exponent = 3)}$$
$$B = 1.0 \times 2^2 \quad \text{(Exponent = 2)}$$

Align exponents:

$$B = 0.5 \times 2^3$$

Now subtract:

$$A - B = (1.0 - 0.5) \times 2^3 = 0.5 \times 2^3 = 4.0$$

Result: 4.0

Note:

These simplified examples use normalized binary representations with perfect rounding for clarity. In actual IEEE-754 hardware, rounding, denormalized numbers, and edge cases like NaNs and infinities are also handled carefully.

IEEE-754: Bit-Level Representation Example

Let us compute $A = 3.5$ and $B = 1.5$, using IEEE-754 32-bit (single precision):

$A = 3.5 = 1.11_2 \times 2^1$
$\Rightarrow$ Sign = 0, Exponent = $127 + 1 = 128 = 10000000_2$
Fraction (remove leading 1) = $110000\ldots0$
$\Rightarrow A =$ `0 10000000 11000000000000000000000`

$B = 1.5 = 1.1_2 \times 2^0$
$\Rightarrow$ Sign = 0, Exponent = $127 + 0 = 127 = 01111111_2$
Fraction = $100000\ldots0$
$\Rightarrow B =$ `0 01111111 10000000000000000000000`

Now perform $A - B$:

- Flip sign bit of $B \rightarrow$ `1 01111111 ...`
- Align exponents by shifting B's fraction
- Add the significands
- Normalize and round

Final result (omitted here for brevity): 2.0 with IEEE-754 encoding `0 10000000 00000000000000000000000`

Special Cases in IEEE-754

Real hardware must handle edge cases:

- **Denormalized numbers**: When the exponent is all zeros, leading 1 is assumed to be 0 instead of 1. Helps represent values close to zero.
- **Infinity**: Exponent all 1s, fraction all 0s. ∞, $-\infty$
- **NaN (Not a Number)**: Exponent all 1s, fraction non-zero. Used for undefined operations like 0/0.
- **Rounding modes**: Round-to-nearest-even (default), toward-zero, toward-$\pm\infty$

Real Processors Use IEEE-754 Adder units

Almost every modern CPU (Intel, AMD) and GPU (NVIDIA, AMD) implements arithmetic using dedicated IEEE-754 hardware units.

- **FPU (Floating Point Unit)**: Part of the CPU core, performs IEEE-754 compliant operations.
- **Pipeline and Microcode**: Floating-point instructions are decoded into micro-operations that route through adder/subtractor, multiplier, and normalization logic.
- **NVIDIA GPUs**: CUDA cores implement 32-bit (FP32), 16-bit (FP16), and newer formats like TensorFloat (TF32), and bfloat16 for AI workloads.
- **Optimization**: CPUs use fused multiply-add (FMA) to perform $A \cdot B + C$ with a single rounding.

In most architectures, real-number subtraction is literally:

$$A - B = A + (-B)$$

And internally, the "negate" is often just a flip of the sign bit.

Weird Case: Subtracting Very Close Numbers

$$A = 1.0000001, \quad B = 1.0$$

IEEE-754 must align exponents, subtract significands, and normalize:

$$A - B \approx 0.0000001$$

Depending on precision (single vs double), the result may underflow or lose precision due to rounding. This is why IEEE-754 compliance is critical in scientific computation.

The Minimalist Backbone of Complex Arithmetic in Hardware

Despite the apparent complexity of floating-point arithmetic—with exponent alignment, normalization, and rounding—the underlying mechanisms remain rooted in our foundational acts: addition and comparison. Aligning exponents requires comparison to determine which operand is larger; normalization involves adding or subtracting shifts, effectively adding zeros or adjusting magnitude; rounding decisions are comparisons to the nearest representable value. Even multiplication and division, implemented via repeated additions and conditional comparisons, ultimately bow to these primal operations. Thus, the entire machinery of modern hardware arithmetic faithfully echoes the minimalist truth:

$$\boxed{\text{All Computation} = \text{Addition} + \text{Comparison}}$$