“Module 3: Assignment 1 - Numerical Data Representations”

Harsh Siddhapura

“Dr. Dinesh Sthapit”

“September 07, 2023”

P1

1. Unsigned numbers, which represent non-negative whole numbers, are a key notion in computer systems. They are widely employed in computer programming and digital logic to accomplish a wide range of arithmetic and logical operations. Unsigned numbers, in contrast to signed numbers, which may represent both positive and negative values, only represent positive values (including zero).

   Unsigned integers are frequently represented in binary representation in computer systems. Starting with the rightmost digit, which represents a power of 2, the following digit to the left indicates a power of 2, the next one represents a power of 2, and so on.

   Here's how unsigned integers are represented in binary format:

   - Start with the decimal (base-10) number you want to represent as an unsigned integer.
   - Divide the decimal number by 2 and record the remainder.
   - Keep on dividing the result you got in the previous step by 2, and keep track of the remainders every time, until you eventually reach a quotient of 0.
   - Write down the remainders in reverse order to obtain the binary representation.

   Let's say we want to represent the decimal number 42 as an unsigned 8-bit binary number.

   - Start with 42. Divide:
     - 42 by 2: 42 ÷ 2 = 21, rem is 0.
     - 21 by 2: 21 ÷ 2 = 10, rem is 1.
     - 10 by 2: 10 ÷ 2 = 5, rem is 0.
     - 5 by 2: 5 ÷ 2 = 2, rem is 1.
     - 2 by 2: 2 ÷ 2 = 1, rem is 0.
     - 1 by 2: 1 ÷ 2 = 0, rem is 1.

   We should now record the leftovers in turn around request: 101010.

   To address this as an 8-bit twofold number, we'll use driving zeros: 00101010.

As a result, the decimal number 42 may be represented in an unsigned 8-bit binary as 00101010.

Unsigned whole numbers are commonly used in computer programmes for tasks like counting, sorting exhibits, and addressing non-negative quantities.

2. Binary representation and binary coded decimal (BCD) are two non-conversion ways for converting numbers to numbers. Let's look at how the values of these two kinds differ and why BCD is useful in particular commercial applications.

- Binary Representation:
    - In binary representation, numbers are represented using base-2 digits (0 and 1).
    - Each digit in a bin number represents a power of 2.
    - The range of values that can be represented in binary is from 0 to (2^n - 1), where n is the number of bits used.
    - For example, in an 8-bit binary representation, the range is 0 to 255 (2^8 - 1).
- Binary-Coded Decimal (BCD):
    - BCD is a type of binary encoding that represents each decimal digit using a 4-bit binary code.
    - In BCD, each decimal digit is encoded individually, and there is no direct relationship between the binary values of adjacent digits.
    - BCD can represent decimal values from 0 to 9 for each digit.

Now, let's discuss why BCD is preferred in certain business applications:

- Decimal accuracy is critical since the bulk of business applications require financial computations. Binary representation can contaminate the usage of numbers in general. BCD, on the other hand, refers to an instantaneous decimal number that regulates the amount of digits without changing.

- Human readability: Binary code is harder to understand than BCD. People may easily detect and decipher mathematical information since each number in BCD documentation is directly related with a decimal number.
- BCD may be translated to and from integers without the need of the binary matching mechanism. This is useful for tasks such as printing numbers on solicitations, receipts, and reports.

As a result, when storage space is limited, binary representation will be favored. However, BCD is still a possibility for applications that need precision, human readability, and arithmetic compatibility.

3. Despite its advantages in commercial applications, BCD takes up less space than binary encoding since each digit is represented by four parts. As a result, when storage space is limited, binary representation will be favored. However, BCD is still a possibility for applications that need precision, human readability, and arithmetic compatibility.

Here's how packed decimal works:

- Each decimal digit (0-9) is represented using a 4-bit binary code (a nibble).
- Packed decimal stores these nibbles consecutively, with no extra bits or padding.
- The leftmost digit can be a sign digit, remaining digits represent the magnitude of the number.

For example, consider the representation of the decimal number +123.45 in packed decimal:

- The sign digit (+) is stored as a 4-bit nibble.
- The remaining digits (12345) are each stored as a separate 4-bit nibble.

In packed decimal, negative numbers are typically represented using a sign digit of 'D' or 'F' for minus, and 'C' or 'A' for plus, depending on the specific encoding scheme.

Common programming languages and computer architectures that support packed decimal include:

- COBOL: It is a programming language widely used in the corporate world. It features “built-in support for crammed decimal data design, making it perfect for monetary and mathematical estimations” (Irv Englander, 2020).
- IBM Servers that are centralized: Local assistance for numeric information fields is available on IBM centralized computers (such as the z/Engineering series). They are commonly used in large-scale financial and business applications.
- PL/I: Programming Language One (PL/I) is a flexible language that can handle a variety of data kinds and more. It is used for a variety of purposes, including monetary and commercial ones.

In these cases, packed decimal is chosen because it allows for the representation and arithmetic of numbers without the errors that can occur when using binary floating point formats. The importance of values in accounting is extremely beneficial. Furthermore, encoded data may be easily converted to and from understandable code, making it an appropriate choice for applications that need to collaborate with humans or outside frameworks in computerized creation.

P2

4. In binary notation, there are different methods for representing the sign of a number, each with its advantages and disadvantages. Let's discuss three common sign representation methods: sign-and-magnitude, 9's decimal complementary representation, and 10's decimal complementary representation.

- Sign-and-Magnitude Representation:
    - MSB, is a tiny bit that represents the sign of a number in both symbolic and numerical form.
    - With MSB set to zero, the number is considered certain and handles the excess extent.
    - If MSB is 1, the remainder, which reflects the dimension as a positive integer, is handled as a negative number.

- This documentation allows basic number expansion and deduction, but it contains two zero representations (+0 and - 0), which can make some jobs difficult.

Example:

- +42 is represented as 00101010 in 8-bit sign-and-magnitude notation.
- -42 is represented as 10101010 in 8-bit sign-and-magnitude notation.

- 9's Complement:
  - The number's sign is represented by the most significant (rightmost) digit, and the magnitude is represented by the magnitude plus the number's complement, which is 9.
  - Positive numbers have a zero prefix and directly express their magnitude.
  - Negative numbers are one, and their magnitude is conveyed as nine or greater.
  - We use standard math to add or subtract numbers, and if there is an exchange from the most noteworthy worth, add it back to the least significant worth (ignore any exchange from the farthest right number).

Example:

- +42 is represented as 0042 in 9's complement notation.
- -42 is represented as 9958 in 9's complement notation (9's complement of 0042).

- 10's Decimal Complementary Representation (Also Known as 10's Complement):
  - The most significant digit is used to indicate the sign in 10's complement format, while the magnitude is stored as the 10's complement of the magnitude, much like in 9's complement representation.
  - A positive number's primary digit is 0 and its magnificence is stated clearly.
  - The magnitude of a negative number is equal to 10 times the prime number 1.

- In this documentation, expansion and subtraction are performed similarly to 9's supplement, but using base 10 integers.

Example:

- +42 is represented as 0042 in 10's complement notation.
- -42 is represented as 9958 in 10's complement notation (10's complement of 0042).

It is vital to remember that the complements 9 and 10 represent the two zeros seen in symbols and numerals. Regardless, they require additional math to accomplish tasks as well as expansion and deduction. Two more representations are extensively employed in modern computer systems due to their simplicity and comfort in math, making these representations unusual.

5. Sign-and-Magnitude, 1's and 2's Complement are three different methods for representing signed numbers in binary notation. Here's a comparison of these representations:

- Sign-and-Magnitude Representation:
  - Advantages:
    - The sign bit directly indicates sign of the number (0 for positive, 1 for negative).
    - Straightforward to convert to and from decimal for human readability.
    - Clear distinction between positive and negative zero.
  - Disadvantages
    - Arithmetic operations (addition and subtraction) are more complex. Special rules are needed to handle sign and magnitude separately.
    - Dual representation of zero can lead to complications in some operations.
    - Not space-efficient as it uses one bit for sign information.
- 1's Complement Representation:
  - Advantages:
    - Simplicity in negating a number (invert all bits to change sign).

        - Arithmetic operations like addition and subtraction are simplified compared to sign-and-magnitude because there's no need to consider sign separately.
    - Disadvantages:
        - Dual representation of zero (positive zero and negative zero).
        - End-around carry must be accounted for in some arithmetic operations.
        - Not as intuitive for humans as sign-and-magnitude.
- 2's Complement Representation:
    - Advantages:
        - Simplifies arithmetic operations further compared to 1's complement.
        - Single representation for zero.
        - Easily obtain the negative of a number by taking the 2's complement.
        - Widely used in modern computer systems due to its computational advantages.
    - Disadvantages:
        - Less intuitive for humans compared to sign-and-magnitude.
        - Overflow detection and handling can be more complex.

In summary:

- Sign and aspect are simple for humans, but difficult for computer science, particularly extension and deduction.
- The 1's complement introduces a double representation, which simplifies arithmetic but should only be used on the terminal.
- The supplement 2 works on math, features a zero digit, and is commonly used in modern PCs.

The specific needs of the computer architecture, as well as the balance between human comprehension and computing speed, typically impact the representation chosen. Because of its efficiency and ability to cope with complex maths, the number 2 has become an important depiction in modern PCs.

P3

6. Complementary representation is a binary notation approach for representing signed integers. It comes in two flavors: radix complement and decreased radix complement. Both approaches have unique uses and qualities, which we will discuss further below:

- Radix Complement (n's Complement): The complement of a number is taken with regard to the base (radix) of the number system being used in radix complement. For binary numbers, this involves determining the 1's or 2's complement.
    - 1's addition (n = 1): Rearrange (reverse) all of the elements in 1's supplement to find the supplement of a parallel number. As a result, each 0 becomes a 1, each 1 becomes a 0.

      For example, 01010101 is the 1's supplement of 10101010.

    - 2nd complement (n = 2): 2's complement is widely used in computations. To investigate the 2's supplement of a double number, first investigate the 1's supplement and then add 1 to the most unimportant component (LSB) of the result.

      Example, the 2's complement of 10101010 is 01010111 (1's complement: 01010101 + 1).

    ➔ Advantages of Radix Complement:
        - ◆ Simplifies arithmetic operations for both addition and subtraction, as you can ignore signs and simply perform binary addition.
        - ◆ Has only one representation for zero.
        - ◆ Easy to obtain the negative of a number by taking the complement and adding 1.

    ➔ Disadvantages of Radix Complement:
        - ◆ In the 1's complement, there are two representations for zero (positive zero and negative zero).
        - ◆ In the 2's complement, the most significant bit (MSB) represents the sign, which can be less intuitive for humans.

- Diminished Radix Complement (n - 1's Complement): Diminished radix complement, also known as the (n - 1)'s complement, is a variant in which the complement is taken with regard to

(n - 1), where 'n' is the number system's base or radix. The reduced radix complement is the 1's complement in binary.

- The 1's Complement (n = 2) is produced by flipping all the bits in binary, much as the 1's complement of the radix complement.
- Advantages and Disadvantages: If you examine the 1's complement (complement) or the 2's complement (reduced radix complement), the advantages and disadvantages of diminished radix complement are largely the same as those of radix complement.

In rundown, expanded documentation (radix's supplement and less radix's supplement) allows a specific number to be referenced twice. Because of its “unique representation of zero and simplicity of computing, the 2's complement form is the most often used in today's computers” (Irv Englander, 2020). However, the 1's supplement structure has a linked history and is occasionally used in exceptional contexts.

7. A way for expressing signed decimal numbers is the 9's decimal complement. It entails determining the complement of a decimal value with the digit 9. This method is very beneficial for simplifying subtraction operations by turning them to addition operations. It's often used in financial computations and on older computers.

Here's the step-by-step process:

- Begin with the decimal number whose 9's complement you want to find.
- Subtract the number's digits from 9.
- Record the findings as the 9's complement digits.
- If the original number was negative, add 1 to the 9's complement's rightmost digit.
- Disregard any carry-out from the leftmost digit (the most significant digit).

Example: Calculating the 9's complement of -3156:

- Start with the decimal number -3156.

- Subtract each digit from 9:
    - 9 - (-3) = 12 (carry)
    - 9 - 1 = 8
    - 9 - 5 = 4
    - 9 - 6 = 3
- Record the results: 8433.
- Given that -3156 is negative, add 1 to the rightmost digit: 8434.
- There's a carry-out of 1 from the leftmost digit (12 - 9 = 3), but we ignore it.
- Therefore, the 9's complement of -3156 is 8434.

The binary complement of the number one, sometimes known as the "complement," is a method for encoding signed binary integers. It entails flipping (inverting) all of a binary number's bits. The major function of the 1's complement, which is typically present in older computer systems, is to simplify subtraction operations in binary arithmetic.

- Start with a binary number for which you want to find the 1's complement.
- Flip (invert) each bit in the binary number. Change every 0 to 1 and every 1 to 0.

Example: Finding the 1's complement of 101110:

- Start with the binary number 101110.
- Flip each bit:
    - 1 to 0
    - 0 to 1
    - 1 to 0
    - 1 to 0
    - 1 to 0
    - 0 to 1
- The 1's complement of 101110 is 010001.

In rundown, the decimal number serves ten and nine, whereas the paired number serves two and one. In either way, the complement of a number must be computed, generally by altering a few bits or digits to make the numbers easier to subtract.

8. Because of the complexities of sign items and specific requirements for activity, computational computations for picture and code encodings might be difficult. For a number of reasons, symbols and numerical encodings employ one bit to indicate the sign (0 for positive, 1 for negative) and another bit to represent the value. As a result, arithmetic and hardware implementation may be hampered:

- Display as an accessible sign Spread occurs during abundance number juggling activities from the plentifulness component to the sign piece as well as the other way around. When working with a big number of maths terms, it might be difficult to keep track of them all.
- Exams utilising pictures and mathematical representations are difficult. When comparing two integers, size and sign should be taken into account.
- Exams utilising pictures and mathematical representations are difficult. When comparing two integers, size and sign should be taken into account.

Converting numbers between binary and other representations, such as sign and magnitude, may be difficult. Many people employ more efficient representations, such as the sum of two, since they make arithmetic easier, decrease hardware complexity, and make dealing with zeros and negative integers easier.

P4

9. Positive and negative integers can be added in 9-digit addition utilising modulo addition and wrapper addition. The number 9 was added. This is how signed numbers can be expressed. He had to add the negative nine numbers together to obtain the sum of the negative nine numbers.

- Modular Addition (Modulo 10 Addition):

- In modular addition, you add two numbers together as you normally would, digit by digit, without considering carries.
- If the sum of any digit pair exceeds 9, you subtract 10 from that sum to bring it within the range of 0-9.
- For example, consider the addition of two numbers, A and B:
  - A = 4831
  - B = 7264
- Digit-wise addition:
  - 1st digit: 4 + 7 = 11 (since it exceeds 9, subtract 10) = 1
  - 2nd digit: 8 + 2 = 10 (subtract 10) = 0
  - 3rd digit: 3 + 6 = 9
  - 4th digit: 1 + 4 = 5
- The result of modular addition is 5095.
- Handling negative numbers: To add a negative number, you first find its 9's complement and then apply modular addition. For example, to add -7264 to 4831:
  - Find the 9's complement of 7264, which is 2735.
  - Perform modular addition on 4831 and 2735 as shown above.
  - The result will be the addition of 4831 and -7264, which is -2433.

- Addition with Wraparound (Carry Around):
  - In addition with wraparound, you perform modular addition just like in the modular addition method.
  - However, if a carry is generated when adding the most significant digit (leftmost digit), you "wrap" it around and add it to the least significant digit (rightmost digit).
  - Handling negative numbers: Similar to modular addition, you first find the 9's complement of negative numbers and then apply addition with wraparound. If a carry is generated during wraparound, it's added to the least significant digit.

- Example with -7264 and 4831:
  - Find the 9's complement of 7264, which is 2735.
  - Perform modular addition on 4831 and 2735 as shown in the modular addition method.
  - If a carry is generated, wrap it around and add it to the least significant digit.
  - The result will be the addition of 4831 and -7264, which is -2433.

There are two approaches to solve negative numbers: To begin, determine the 9's supplement of the negative number, and then do the expansion. “The major distinction between isolated expansion and helical expansion is how they are expressed: Modulo expansion subtracts ten from the convey, whereas helical expansion wraps the convey to an indivisible number” (Irv Englander, 2020).

10. Subtracting in two-addition arithmetic entails adding another number to make it easier. Extra activities include nine or more and 10 in addition to two. I'll explain deduction using 10s and a model in the section below.

Subtraction of the complement of ten: In 10's complement arithmetic, the same integers from 0 to 9 are used to represent positive and negative numbers. The deduction using 10's supplement may be divided into many stages:

- Find the 10's Complement of the Number to be Subtracted:
  - To subtract a number (let's call it B) from another number (A), first find the 10's complement of B. This involves subtracting each digit of B from 9 (the largest digit in the decimal system).
  - The 10's complement of B is often represented as $\overline{B}$.
- Add the 10's Complement to the First Number:
  - Add the 10's complement of B ($\overline{B}$) to the first number (A).
- Check for a Carry:
  - If a carry is generated in the leftmost digit of the addition, ignore it.

- The Result:
    - The result of this addition is the subtraction of B from A.

Example: Subtract 327 from 548 using 10's complement arithmetic:

- Find the 10's Complement of 327 ($\overline{B}$):
    - Subtract each digit of 327 from 9:
        - 9 - 3 = 6
        - 9 - 2 = 7
        - 9 - 7 = 2
    - The 10's complement of 327 is 672.
- Add the 10's Complement to 548:
    - Add 672 to 548:

      548 +672 = 1220

- Check for a Carry:
    - In this case, there is no carry.
- The Result:
    - The result of subtracting 327 from 548 using 10's complement arithmetic is 1220.

By changing the subtraction issue into an addition problem, this technique successfully replicates subtraction. We accomplish consistent subtraction for both positive and negative integers by determining the 10's complement of the number to be subtracted and adding it to the first number.

11. Overflow is a common issue in computer arithmetic when the result of a calculation is too large to fit within the available storage space, which is often a fixed number of bits. It's like trying to pour too much water into a small cup; it spills over, and we need to figure out how to handle it.

Now, let's talk about how we deal with overflow using two different methods:

- End-Around Carry in Modular Arithmetic: Think of modular arithmetic like a clock with limited hours (e.g., 1 to 12). When we add or subtract numbers and go beyond 12, we don't keep counting endlessly; we go back to 1 and start over. This is called "end-around carry."
    - For example, if we add 11 and 4 in a clock-like system, we'd ideally get 15. However, since our clock only goes up to 12, we wrap around, the result becomes 3 (15 - 12 = 3).
    - End-around carry prevents numbers from getting too big and ensures that our calculations stay within the limited range. But keep in mind that it might lose some precision in the process.
- Complementary Arithmetic: In complementary arithmetic, we use a clever trick to represent both positive and negative numbers with a fixed number of digits.
    - For positive numbers, overflow is handled similarly to modular arithmetic. If the result is too big to fit, it wraps around, and we ignore the extra part.
    - But for negative numbers, we do something special. When a negative result overflows, it still wraps around, but we also add a "borrow." This borrow tells us that the result is negative, and we adjust the magnitude of the number accordingly.

Consider four integers in addition to nine: 9999 (best value), and if we add one, we should obtain 10000. However, because we only have four numbers, we must modify it to 0000. However, a negative 1 is also appended to signify. The result is a negative number, -999.

When it is vital to illustrate whether “the results of two computations are positive or negative, we guarantee that our estimations are conducted sequentially” (Irv Englander, 2020).

Finally, computer arithmetic overflow happens when we uncover an excessive number of possibilities. While the terminal contains clock-like objects in arithmetic, two-point arithmetic switches translate numbers to indicate whether the value is positive or negative within storage constraints. This approach makes our computations more manageable and efficient.

P5

12. A twofold number in double representation can have 1 added to it by simply rearranging (fixing) all of the components in the initial number. This means that each 0 becomes a 1 and each 1 becomes a 0. Because you are completing 1s, this is referred to as a "1's complement." This is mostly used for deduction in more established computer frameworks.

Take the binary number 101101 for example: Reversing each digit yields the sum 1:010010. The number is expressed as an increase of one by taking one digit away from the number 9. The following is an example of how to compute the 4567 to 1 ratio:

- 9 - 4 = 5
- 9 - 5 = 4
- 9 - 6 = 3
- 9 - 7 = 2

So, the 1's complement of 4567 is 5432.

10's supplement (10's supplement): According to parallel documentation, adding a twofold number by 10 is similar to adding 1, but with an additional step. To begin, reorder all of the pieces as though you were inverting the 1's supplement. Then, at that moment, multiply the result by 1. This is referred to as "terminal expulsion" in addition to 1. In earlier computer systems, this notation, like the addition of one, is used for subtraction.

For example, consider the binary number 101101:

- Flip each bit: 010010.
- Add 1 to the result: 010010 + 1 = 010011.

When calculating the 10's complement in decimal form, each number is subtracted from 9 as with the 1's complement before the result is multiplied by 1. For example:

- 9 - 4 = 5

- 9 - 5 = 4
- 9 - 6 = 3
- 9 - 7 = 2

Now, add 1 to the result:

- 5 + 1 = 6
- 4 + 1 = 5
- 3 + 1 = 4
- 2 + 1 = 3

So, the 10's complement of 4567 is 5433.

Both 1's complement and 10's complement representations are used for subtraction operations, and they allow subtraction to be performed using addition with some adjustments. The choice between them depends on the specific system and the preferences of the designers.

13. To pick between 1's complement and 10's complement, you need to consider what suits your specific computing needs:

- 1's Complement: It's simple to grasp but has the quirk of dual zero representations and needs extra steps for end-around carry. Older computer systems used it, but it's not as common today.
- 10's Complement: It provides a single way to represent zero and simplifies arithmetic by making subtraction equal to addition. It does, however, need an additional bit for end-around carry and has a slightly more difficult negation mechanism. It's also from the past and isn't as common in current computers.

Advantages of 1's Complement:

- Simplicity: Working with 1's complement is quite simple. You just flip all the bits in the number, and that's it.

- Easy Negation: When you want to turn a number negative in 1's complement, it's a breeze. You just flip the bits again, and you're good to go.

Disadvantages of 1's Complement:

- Double-Zero Trouble: Here's the tricky part – 1's complement has two ways to represent zero: an all-0 pattern and an all-1 pattern. This can make things confusing when you're doing math.
- Dealing with Carry: Subtraction using 1's complement needs a separate step to deal with what's called "end-around carry," which adds a touch of complexity.

Advantages of 10's Complement:

- Single Zero: Unlike 1's complement, there's only one way to represent zero in 10's complement. This simplifies math and avoids any confusion.
- Subtracting is Adding: When you're subtracting using 10's complement, it's basically the same as adding. It's super straightforward.

Disadvantages of 10's Complement:

- Extra Bit: 10's complement requires an extra bit to handle "end-around carry," which means you need more space in your storage.
- Tricky Negation: To make a number negative in 10's complement, you need to do a bit-flip plus add 1, which is a tad more involved than the simple flipping in 1's complement.

These days, most computers prefer two's complement because it's efficient and straightforward for both addition and subtraction. Still, you might encounter 1's complement and 10's complement in older or educational systems.

14. Converting a number from 9's complement to 1's complement is like a two-step dance:

- Step 1: We start by finding the 10's complement of the 9's complement. Think of it as a game of "subtract from 9."

- So, for our number 472 in 9's complement, we do the math:
  - 9 - 4 = 5
  - 9 - 7 = 2
  - 9 - 2 = 7
- The result is 527. That's our 10's complement.

- Step 2: Now, we add 1 to our 10's complement to get the 1's complement. It's like adding a cherry on top.
  - We take 527 and simply add 1:
    - 527 + 1 = 528
  - The 1's complement of 472 in 9's complement is 528.

We can reverse the process and check if we get back to the original number:

- Step 1: Find the 10's complement of 528 in 9's complement. Subtract from 9:
  - 9 - 5 = 4
  - 9 - 2 = 7
  - 9 - 8 = 1
  - We have 471 as the 10's complement.
- Step 2: Add 1 to our 10's complement:
  - 471 + 1 = 472

We're back to our original number, 472. So, the conversion from 9's complement to 1's complement checks out.

P6

15. Exponential notation, also known as scientific notation, is like shorthand for expressing really big or really tiny numbers. It's a handy way to deal with these numbers without writing out a ton of digits. In this special notation, you write a number as two parts: a coefficient and a power of 10. It looks like this:

a x 10^n

- a: This is coefficient, usually a number between 1 & 10. It holds important figures of numbers.
- n: The exponent shows how many times the coefficient is multiplied by 10 in the equation.

Exponential Notation matters because of following:

- Keeping Things Consistent: Scientists and engineers love exponential notation because it's the same everywhere. It keeps scientific data consistent and avoids confusion when communicating big ideas.
- Math Made Easy: Doing math with exponential notation is a breeze. When you multiply or divide numbers like this, you just add or subtract their exponents. It's way less messy than dealing with long strings of numbers.
- Getting a Feel for Scale: Exponential notation helps you quickly grasp how big or small a number is. Just look at the exponent, and you'll know if you're dealing with thousands, millions, billions, and so on, without counting zeros.

In a nutshell, exponential notation is like a secret code for handling both enormous and tiny numbers. It's a handy tool for making math less messy, keeping science clear, and helping us understand the size of things in the scientific world.

16. The IEEE 754 standard is like the rulebook computers use to handle numbers that aren't nice and neat whole numbers. It helps computers represent real-world numbers with decimals, like what you'd use in science and engineering. Parts of the 32-bit IEEE 754 Floating-Point Format:

- The one-bit Sign Bit: Envision the furthest left piece as a small banner. The number is positive, like a green light, if it is zero. In the event that it's 1, it implies the number is negative, similar to a red light.
- The Eight-Bit Exponent: Consider the following eight bits to be your power. These pieces address the type, which lets you know how frequently you'll duplicate your fundamental number

(we'll get to that in a little) by 2. These bits are stored in a way that makes math work well, so they have a special trick. By 127, they are "biased." Therefore, stating that the power is -126 is equivalent to stating that all of the bits in this array are 0s; It is equivalent to stating the power is +127 if they are all 1s.

- The Significant (Mantissa) (23 pieces): Presently, the other pieces (23 of them) resemble the digits after the decimal point in a decimal number. “They hold the part of the number that isn't a whole” (Irv Englander, 2020). This part can be considered a portion somewhere in the range of 1 and 2, and it's made to fit pleasantly with the example.

Example: Represent the number 6.75 using this format:

- Sign Bit: Positive number, so the flag is 0.
- Exponent: To deal with 6.75, we need to write it in a certain way, like 1.1011 x 2^2. So, the exponent is 2 (adjusted to 129 in binary, which is 10000001).
- Significand: The fraction part is 0.1011. But, we don't write the first 1 because we assume it's always there. So, it's 01100000000000000000000.

Now, we put it all together:

- Sign Bit: 0 (positive)
- Exponent: 10000001 (adjusted)
- Significand: 01100000000000000000000

The representation of 6.75 becomes 01000000110110000000000000000000.

This format is like a universal language in which computers can deal with any number, whether it is a number or a large number. This is very important in science and engineering, where precision is important.

17. There are various advantages of source representation such as:

- Consider floating point to be a method for altering numbers using numbers. “It is useful in industries like as science and finance where precision with numbers is required, even if the quantities are modest” (Irv Englander, 2020).
- Major and minor: A floating point can be any number from the lowest to the greatest. Many computations in science and engineering can benefit from this.
- Error correction: Measurements and computations are not always exact. Floating point can handle these techniques and improve things.

Notes and issues when working with floating point:

- Not very specific: It's important to remember that floating point isn't always perfect. Changing the numbers might sometimes result in minor inaccuracies in computations.
- Hard comparisons: Because to this problem, comparing two dot numbers to check if they are the same may be a little easier. “To establish if they are close enough, you must normally specify certain limitations” (Irv Englander, 2020).
- Some numbers, such as 1/3 or 1/7, cannot be completely represented as floating point numbers. As a result, minor mistakes may arise while doing these computations.

Compare with written numbers:

- Set the numbers to be tallied to be compared. They are useful for counting and need less memory than floating point numbers for the same number.
- Notation for floating points: Floating point is a master of real numbers - the number-filled universe of numbers. It is useful in science and engineering, albeit it has certain limits in terms of precision and balance.

Finally, choose between floating point and number according to your needs. In research and engineering, floating point is used to increase the precision of real values. When tasks like “calculating or working with numbers require precision and efficiency, stick to numbers” (Irv Englander, 2020). Most programmes employ both modes to balance accuracy and efficiency.

References

Irv Englander. (2020). *ARCHITECTURE OF COMPUTER HARDWARE, SYSTEMS SOFTWARE, AND NETWORKING : an information... technology approach.* John Wiley.