The Complement Approach to Subtraction Using Addition in Computer Number Systems

A. Tarek\textsuperscript{1}, and A. Alveed\textsuperscript{2}

\textsuperscript{1}Engineering, Physical and Computer Sciences, Montgomery College, Rockville, Maryland, U.S.A.
\textsuperscript{2}Owen J. Roberts High School, Pottstown, Pennsylvania, U.S.A.

Abstract—Computer stores negative numbers in 2’s complement form as addition is faster, and cheaper compared to subtraction. The processor and other major hardware circuitry implement the complement-based addition for subtraction. Besides, adder hardware is considerably cheaper than subtractor. Hence, computers would rather add negative numbers in their complement form to a positive number instead of performing a direct subtraction to optimize resources. There are four prevalent number systems in computing. These are binary, decimal, octal, and hexadecimal number systems. In this paper, it is demonstrated that the same uniform complement approach may be used across all four number systems to perform subtraction using addition. For simplicity and uniformity, the complement approach presented is limited to 8-bit Binary, and 8-digit Decimal, Octal, and Hexadecimal numbers. The complement to the base approach is capable of producing the correct result for a specific data size (memory size). The approach presented may be extended to any \textit{n}-bit or \textit{n}-digit data size used for actual computation, where \textit{n} is a positive integer.

Keywords: Complement of base, Binary Number System, Decimal Number System, Octal Number System, Hexadecimal Number System, Adder hardware

1. Introduction

There are four well-known number systems in computing. Addition-based subtraction demands particular interest in computer number systems, as adder circuitry is cheaper, easier to design, and faster compared to the subtractor circuitry. This has drawn special interest to the design of the Central Processing Unit (CPU) hardware, as speed is of paramount importance here. Also, adder circuit may be used to implement multiplication hardware, and subtractor circuits may be used to implement division hardware. Since the adder circuit could be used to implement subtractor, therefore, the adder circuit may also be used to implement division hardware. In this original paper, we explore the addition-based subtraction in the four prevalent computer number systems. In fact, the binary number system with 2’s complement approach forms foundation to the hardware implementation of the addition-based subtraction in CPU circuitry. The other three number systems, such as the Decimal, the Octal, and the Hexadecimal number systems may be used for the higher level analysis of the addition-based subtraction. In this paper, the same uniform addition-based subtraction example problem is considered across all four number systems for consistency, and it is demonstrated that the addition-based subtraction technique equally applies to all four computer number systems.

Following forms the basis of analysis in this paper. Any sequence of \textit{n} digits, \(a_{(n-1)}a_{(n-2)}\ldots a_1a_0\) in any one of the four computer number systems is interpreted as an unsigned integer \(N\), whose value is given by, \(N = \sum_{j=0}^{(n-1)} b/a_j\), where \(b\) is the base to the number system. Here, \(b \in \{2, 8, 10, 16\}\) for the binary, octal, decimal, and hexadecimal number system, respectively.

The number of bits or binary digits to represent data in main memory or the Random Access Memory (RAM), in case of Personal Computer (PC), is important in addition-based subtraction of binary numbers and also for other three computer number systems. The CPU fetches data from the main memory or RAM, and stores the result of the computation to the main memory or RAM as well. The number of bits to represent data eventually translates to the number of digits to represent data in decimal, hexadecimal, and octal number systems. For instance, with 8-bit binary, one can represent a maximum of 256 different numbers in decimal beginning with 0 up to 255. So, it is necessary to have 3 digits in decimal to represent the binary data that are up to 8-bits long. However, for the uniformity of analysis in this paper, only 8-bit Binary numbers, and 8-digit Decimal, Octal, and Hexadecimal numbers are being dealt with. Since only 8 bit / 8 digit numbers are being considered in this paper, to avoid any overflow or underflow error, a number is subtracted from another positive number, instead of adding two positive numbers or two negative numbers. Therefore, all throughout the paper, one positive number is paired with a negative number, since always, adding a positive number to a negative number produces a resulting number, whose magnitude is smaller than the larger of the two original numbers.

The base’s complement-based approach is a powerful computational tool. The technique may easily be extended to multiplication and division arithmetic. For multiplication, if the multiplicand and the multiplier are each \(n\) bits or \(n\) digits long, where \(n \in \mathbb{Z}^+\), each one of them is required to be extended to twice the original size, which is \(2n\) bits or \(2n\) digits. This is required to be done before the complement approach to multiplication may be applied. This implies adding sufficient numbers of 0s to the left of a positive number, and sufficient numbers of highest digit \(ms\) to the left of a negative number expressed in the base’s complement form in one of the four computer.
number systems. Here, \( m \in \{1, 7, 9, F\} \).

In Section 2, specific terms and notations used in this paper are discussed briefly. Section 3 explores combinatorial results based on the addition-based subtraction technique explored through the base \( b \)'s complement approach. Section 4 deals with the 2's complement-based addition to perform subtraction in Binary Number System. Section 5 discusses the Decimal Number System with base 10 to perform base's complement-based addition instead of direct subtraction. Section 6 deals with the Octal Number System for the base 8's complement-based addition for subtraction. Section 7 is the addition-based subtraction in Hexadecimal number system. Section 8 considers conclusions based on the results, and analysis throughout the paper, and explores future research avenues.

2. Terminology and Notations

Throughout the paper, following notations are used.
\( b \): The base of a computer number system. Here, \( b \in \{2, 8, 10, 16\} \).
\( a = a_1a_2a_3a_4a_5a_6a_7a_8 \): A 8-bit or a 8-digit number in one of the four computer number systems. Here, for \( i \in 1, 2, 3, \ldots, 8 \), \( a_i \in \{0, 1\} \) (for binary number system)
\( a \in \{0, 1, 2, \ldots, 9\} \) (for decimal number system)
\( a = a_1a_2a_3a_4a_5a_6a_7a_8 \): A 8-digit number (for octal number system)
\( a \in \{0, 1, 2, \ldots, F\} \) (for hexadecimal number system).

If the actual number of bits or digits, \( n \), in the given number \( a \) in one of the four number systems is less than \( 8 \), then \( 0 \)'s are added to the left of the number, \( a \) to make it either a 8-bit or a 8-digit number. Adding \( 0 \)'s to the left of a number does not change the numerical value of the number.

\( mmmmmmmmm \): The largest 8-bit or the 8-digit number in one of the 4 computer number systems. Here, \( m \in \{1, 7, 9, F(\text{hexdigit})\} \).
\( m \)'s Complement of \( a \): The \( m \)'s Complement in 8 bit or in 8 digit in one of the 4 number systems under considerations is given by, \( mmmmmmmmm - a = mmmmmmmmm - a_{1}a_{2}a_{3}a_{4}a_{5}a_{6}a_{7}a_{8} \). Here, \( m \in \{1, 7, 9, F(\text{hexdigit})\} \).
Base \( b \)'s Complement: The base, \( b \)'s complement in 8-bit or 8-digit in one of the 4 computer number systems under considerations is given by, \( mmmmmmmmm - a + 1 = mmmmmmmmm - a_{1}a_{2}a_{3}a_{4}a_{5}a_{6}a_{7}a_{8} + 1 = (b^8 - a) = b^8 - a_{1}a_{2}a_{3}a_{4}a_{5}a_{6}a_{7}a_{8} \). Here, \( b \in \{2, 8, 10, 16\} \).

Highest Digit Complement of a Number System Digit:
Let \( m \) be the highest digit in one of the four computer number systems. Here, \( m \in \{1, 7, 9, F(\text{hexdigit})\} \). Let \( d \) represents a digit in one of the 4 number systems, which is the same number system as of \( m \) under consideration. Then the Highest Digit Complement of \( d \) in the number system under consideration is given by, \( m - d \). For example, if \( m = F \), and \( d = 1 \), the highest digit complement of \( d \) in hex is \( E \). If \( d \) is 1 in binary, then the highest digit complement is 0. For \( d = 1 \) in Octal, the highest digit complement is 6. Finally, for \( d = 1 \) in Decimal, the highest digit complement is, \( 9 - 1 = 8 \).

Overflow: An overflow occurs when the result of the computation may not be correctly represented in 8 bit or in 8 digit in the number system under consideration. This is because the computed result is larger than that may be represented with 8 bits or 8 digits in the number system.

Underflow: An underflow occurs when the computed result is too small to fit in within the 8 bits or 8 digits in the number system under consideration.

MSB: MSB is an abbreviation for the Most Significant Bit. It is the left-most bit, or the bit having the highest weight with a multiplying factor of \( 2^{(n-1)} \) in a n-bit representation of a binary number. In this paper, \( n = 8 \).

MSD: MSD is an abbreviation for the Most Significant Digit. It is the left-most digit, or the digit with the highest weight with a multiplying factor of \( b^{(n-1)} \) in a n-digit representation of a number in a computer number system. In this paper, \( n = 8 \), and \( b \in \{8, 10, 16\} \). Binary numbers are treated separately with MSB.

3. Formal Theoretical Analysis

Following are some important results pertaining to the addition-based subtraction in computer number systems.

Lemma 1 (Complement of the Complement to Base):
The base, \( b \)'s complement of a negative number, \(-a\) represented in one of the four computer number systems provides with the original number, \( a \).
Proof: Negative numbers are represented as base, \( b \)'s complement form. So \(-a\) will be represented as, \( b^n - a \). Here, \( n \) is the number of bits or digits used for the complement form. Throughout this paper, \( n = 8 \). Therefore, \(-a\) is represented as, \( b^8 - a \). The base, \( b \)'s complement of \(-a\) is, \( b^8 - (b^8 - a) = b^8 - b^8 + a = a = \text{The original number without sign} \).

Theorem 2 (Model of Computation Theorem): The Addition-based Subtraction Model presented in this paper accurately computes the difference between two \( n \)-bit/\( n \)-digit numbers in one of the four computer number systems. Here, \( n \geq 1 \).
Proof: Suppose, the first \( n \)-bit/\( n \)-digit number is \( a \). The second \( n \)-bit/\( n \)-digit number is \( c \). In this paper, \( n = 8 \). Both the numbers are presented in one of the four available computer number systems having a base of \( b \). It is required to compute \((a - c)\) using the Addition-based Subtraction approach in one of the four computer number systems. As the negative numbers are presented in base, \( b \)'s complement form, therefore, \((a - c) = a + (-c) = a + (b^n - c) = b^n + a - c \). There are three different cases possible.

1. In the first case, \( a > c \). So, \((a - c)\) will produce a positive result. In the expression, \( b^n + a - c \), \( b^n \) will generate a spurious carry from the \( n \)th bit / \( n \)th digit to the \((n + 1)\)th bit / \((n + 1)\)th digit. Discarding the \((n + 1)\)th bit / \((n + 1)\)th digit generated carry, provides with the accurate result of the subtraction, which is \((a - c)\), and is positive.

2. In the second case, \( a = c \). Therefore, \((a - c) = 0 \). In \( b^n + a - c \), \( b^n \) will generate a spurious carry from the \( n \)th bit / \( n \)th digit to the \((n + 1)\)th bit / \((n + 1)\)th digit.
digit. Discarding the \((n + 1)\)th bit / \((n + 1)\)th digit generated carry, will yield with the correct result of subtraction, which in this case is, 0.

3) In the third, and the final case, \(a < c\). The expression, \(b^n + a - c\) becomes \(b^n + (c - a)\). As \(c > a\), therefore, \((c - a)\) is positive, and is represented in the base, \(b\)'s complement form in \((b^n - (c - a))\). So, there will be no generated carry from the \(n\)-th bit / \(n\)-th digit to the \((n + 1)\)th bit / \((n + 1)\)th digit. Instead, there will be a non-zero bit / digit at the \(n\)-th bit / \(n\)-th digit position (MSB / MSD). Using the above Lemma, taking the base, \(b\)'s complement again provides with the negative of the result of the subtraction, which is \((b^n - b^n + (c - a)) = (c - a)\). Adding a negative sign in front of the result obtained provides with the accurate result of subtraction, which is, \(-(c - a) = (a - c)\). Obviously, as \(a < c\), the result of the subtraction using addition will be negative.

Hence, the computational model presented in this paper for all four of the Computer Number Systems accurately computes the difference between two \(n\)-bit/\(n\)-digit numbers (in the paper, \(n = 8\)).

**Theorem 3 (Highest Value Expansion Theorem):** The highest value in \(n\) bit or in \(n\) digit in a Computer Number System is given by, \(\ldots m...m\) \((n\) consecutive \(m\)s\) \(= b^n - 1 = (b - 1)(b^{n-1} + b^{n-2} + \ldots + b + 1)\). Here, \(b\) is the base of the number system under consideration. In this paper, \(n = 8\).

**Proof:** The Theorem will be proved using Mathematical Induction J.

1) Basis: The Basis for the Induction is, \(n = 1\). With \(n = 1\), L.H.S. = \(b^n - 1 = b - 1\). R.H.S. = \((b - 1)(b^{n-1}) = (b - 1)(1)\), and the given hypothesis is true for the basis.

2) Induction: Suppose, the Inductive Hypothesis is, \(b^k - 1 = (b - 1)(b^{k-1} + b^{k-2} + \ldots + b + 1)\), where \(k \geq 1\). It is required to prove that, \(b^{k+1} - 1 = (b - 1)(b^1 + b^{k-1} + b^{k-2} + \ldots + b + 1)\). The R.H.S. = \(b - 1)(b^{k+1} - b^{k} - b^{k-1} - \ldots - b - 1)\). According to the Inductive Hypothesis, \((b - 1)(b^1 + b^{k-1} + b^{k-2} + \ldots + b + 1) = (b - 1)(b^1 + b^{k} + b^{k-1} - b^{k-2} + \ldots + b + 1) = b^{k+1} - 1 = L.H.S.

3) Conclusion: The given Theorem is True for the base case, \(n = 1\). When it is true for \(n = k\), \(k \geq 1\), it is also true for \(n = k + 1\), \(k + 1 \geq 1\). As it is true for \(n = 1\), so it is also true for \(n = 2\). As it is true for \(n = 2\), so it is also true for \(n = 3\). Proceeding this way, it is true in general for any \(n, n \geq 1\).

**Corollary 4 (Complement of the Base):** The base, \(b\)'s complement in \(n\)-bit or in \(n\)-digit of a given number, \(a\) in one of the four computer number systems is given by, \(\sum_{i=0}^{n-1}(b^i)m + 1 - a\). Here, \(m\) is the highest digit in the number system.

**Proof:** Base, \(b\)'s complement in \(n\)-bit or in \(n\)-digit for a number, \(a\) in one of the four computer number systems is given by, \(b^n - a = b^n - 1 + 1 - a\). From the above Theorem, \(b^n - 1 = (b - 1)(b^{n-1} + b^{n-2} + \ldots + b + 1)\). Therefore, \(b^n - a = b^n - 1 + 1 - a = (b - 1)(b^{n-1} + b^{n-2} + \ldots + b + 1) + 1 - a\). Since, \((b - 1) = m = m\) the highest digit in the number system, therefore, \(b^n - a = (m)(b^{n-1} + b^{n-2} + \ldots + b + 1) + 1 - a = \sum_{i=0}^{n-1}(b^i)m + 1 - a\).

**Corollary 5 (Geometric Progression of Base Power):** The sum of the powers of base, \(b\) from 0 through \((n-1)\) adds up to \((\frac{(b^n-1)}{b-1})\) or \((\frac{(b^n-1)}{m})\). Therefore, \(\sum_{i=0}^{n-1}b^i = (\frac{(b^n-1)}{b-1}) = (\frac{(b^n-1)}{m})\).

**Proof:** From the above Theorem, \(b^n - 1 = (b - 1)(b^{n-1} + b^{n-2} + \ldots + b + 1)\). Therefore, \(b^{n-1} + b^{n-2} + \ldots + b + 1 = \frac{(b^n-1)}{b-1}\). But \(b - 1 = m = m\) the largest digit in the computer number system under consideration. Hence, \(b^{n-1} + b^{n-2} + \ldots + b + 1 = \frac{(b^n-1)}{m}\).

**Lemma 6 (Base’s Complement Representation):** The base, \(b\)'s complement representation of both positive and negative numbers is given by, \(N = b^{(n-0)}a_{(n-1)} + \sum_{j=0}^{(n-2)}b^ja_j\) in any one of the four computer number systems.

**Proof:** \(-b^{(n-1)}a_{(n-1)} + \sum_{j=0}^{(n-2)}b^ja_j\) defines the base’s complement representation of both positive and negative numbers. Whenever, the co-efficient \(a_{(n-1)} = 0\), the term \(-b^{(n-1)}a_{(n-1)} = 0\), and the remaining part, \(\sum_{j=0}^{(n-2)}b^ja_j\) defines a nonnegative integer in the number system under consideration. If \(a_{(n-1)} = 1\), for a negative number, the term \(b^{(n-1)}\) is subtracted from the summation to yield a negative number in base, \(b\)'s complement form. Base’s complement representation facilitates the addition and subtraction, the most primitive operations carried out by the CPU. This is the reason, why the base 2’s complement representation is used by the CPU.

**Theorem 7 (Correctness of Complement Form):** The base, \(b\)'s complement of a given number, \(N = \sum_{j=0}^{(n-2)}b^ja_j\) correctly represents the negation of the given number, \(N\) in one of the four computer number systems. Here, \(b \in \{2, 8, 10, 16\}\).

**Proof:** Consider the \(n\)-digit given number, \(N = b^{(n-1)}a_{(n-1)} + \sum_{j=0}^{(n-2)}b^ja_j\). Performing the digit wise complement of the highest digit \(m\) in the given number system, where \(m \in \{1, 7, 9, 15(F)\}\), and adding 1 to the resulting complement value provides with the base’s complement value, \(M\) of \(N\). Now, if \(a_{(n-1)} = 0\), represents the digit-wise complement of \(m\), then treating this as an unsigned integer in the number system, and adding 1 provides with the base’s complement value of \(N\), which is \(M\). Therefore, \(M = b^{(n-1)}a_{(n-1)} + 1 + \sum_{j=0}^{(n-2)}b^ja_j\). Since, \(M\) is the base’s complement representation of \(-N\), hence, \(M + N = 0\). Adding, \(M + N = (a_{(n-1)} + a_{(n-1)})b^{(n-1)} + 1 + (\sum_{j=0}^{(n-2)}b^ja_j) = -m_{(n-1)}b^{(n-1)} + 1 + (\sum_{j=0}^{(n-2)}b^ja_j)\). Here, \(m_j, j \in \{0, 1, 2, \ldots, (n - 1)\}\), represents the highest digit \(m\) at digit/bit position \(j\). Hence, \(M + N = -m_{(n-1)}b^{(n-1)} + 1 + (\sum_{j=0}^{(n-2)}b^ja_j)\). As \(1 + (\sum_{j=0}^{(n-2)}b^ja_j) = m_{(n-1)}b^{(n-1)}\) in any one of the four computer number systems, hence, \(M + N = -m_{(n-1)}b^{(n-1)} + m_{(n-1)}b^{(n-1)}\).
Theorem 8 (No Overflow Or Underflow Assurance): Addition-based Subtraction guarantees that no overflow or underflow error will ever occur.

Proof: As for the addition-based subtraction considered in this paper, one of the two numbers is always positive, and the other one is always negative. So, the result of the computation is always smaller than the largest of the two numbers under consideration. As the largest number fits into a 8 bit or a 8 digit memory, so the generated result will perfectly fit into the 8 bit or the 8 digit memory, and no overflow or underflow error will ever occur.

Theorem 9 (Benefit of Base’s Complement Approach): The base, \( b \)'s complement approach removes the ambiguity of presenting \(+0\) and \(-0\) in \( n \)-bits or in \( n \)-digits using the highest digit, \( m \)'s complement approach in any of the four Computer Number Systems. Using the base, \( b \)'s complement approach, there is only one representation for \(+0\) and \(-0\), which is 0.

Proof: The \( n \)-bit or the \( n \)-digit representation of \(+0\) is 0. The \( n \)-bit or the \( n \)-digit representation on \(-0\) as the highest bit/digit \( m \)'s complement approach in any of the four computer number systems is \( \underbrace{m m m \ldots m}_{n \text{, consecutive } m \text{ bits/digits}} \) (\( n \), consecutive \( m \) bits/digits). Therefore, \(+0\) and \(-0\) have different values in the computer number system. Using the base, \( b \)'s complement approach, the representation of \(-0\) in the computer number system is, \( \underbrace{m m m \ldots m}_{n \text{, consecutive } m \text{ bits/digits}} +1 = 000 \ldots 0 \) (ignoring the \( n \)th bit / \( n \)th digit to the \( (n+1) \)th bit / \( (n+1) \)th digit generated carry). As a result, both \(+0\) and \(-0\) have a unique representation, which is the number 0, removing the ambiguity in the representation of \(+0\) and \(-0\) in the highest bit / digit, \( m \)'s complement approach.

Example 10 (Base’s Complement Approach):
Consider the 9’s complement approach in base 10 with 8 digits. Representation of \(+0\) = 00000000, \(-0\) = 00000000. However, representation of \(-0\) = 00000000. \(-0\) = 00000000. But the representation of \(-0\) in base, 10’s complement approach = (99999999) + 1 = 100000000. Ignoring the 8th digit to the 9th digit generated carry, \(-0\) = 00000000. Therefore, both \(-0\) and \(+0\) have a unique representation, which is the number 0 using the base 10’s complement approach. Same analysis holds for the other 3 number systems as well.

4. 2’rs Complement in 8-bit Binary
Following are the algorithmic steps.

1) Express the number to be subtracted in binary as a 8-bit binary number.
2) Find out the 1’s complement of the binary number to be subtracted by flipping the 0 bits to 1s, and 1s to 0s.
3) Next, find out the 2’rs complement of the binary number by adding 1 to the Least Significant Bit (LSB).
4) Express the number from which to subtract as a 8-bit binary number.
5) Perform addition between the two 8-bit binary numbers.
6) If there is a carry to the 9th bit, simply ignore the generated carry. The math takes care of that. Therefore, the result of the subtraction is positive, given by the remaining 8-bit binary number.
7) Otherwise, if there is a 1 at the Most Significant Bit (MSB) position, or at the left-most bit position, the result of the subtraction is negative. According to the hardware logic, the negative result is now in the 2’rs complement form. Take the 2’rs complement of the generated result one more time by following through the above mentioned steps 1, 2, and 3, and add a negative sign in front of the 2’rs complement obtained, which yields you the negative result generated from the subtraction. The 2’rs complement of the 2’rs complement of a binary number gives you the original binary number, which obviously, is the result of the subtraction with a negative sign in front of it.

If the computation arrives at Step 6, the result of the subtraction is certainly positive. So, discarding the 9th bit (the \((n+1)\)th bit) generated carry provides you with the result of the subtraction. Otherwise, the result of the subtraction is negative, and the computation eventually arrives at Step 7. Following example illustrates the logic underlying the above algorithmic steps.

Example 11 (Base-2’s Complement Example):
Consider 178-212 and 212-178 in decimal. The subtraction will be performed in binary using the addition-based subtraction model. Now, 178\(_{10}\) = 10110010. Also, 212\(_{10}\) = 11010100. Therefore, 178\(_{10}\)-212\(_{10}\) = 178\(_{10}\) + (-212\(_{10}\)) = 10110010 + (-212\(_{10}\)) = 212\(_{10}\) in binary of 212\(_{10}\). However, 212\(_{10}\) = 11010100. Hence, 1’rs complement of 11010100 = 00101011. Now, the 2’rs complement = (00101011 + 1) = 00101100. Next add 10110010 to 00101100. So, 10110010 + 00101100 = 11011110. The Most Significant Bit (MSB) from the result of the addition is 1. So, the result of the subtraction is negative. Taking the 1’s complement of the generated result produces, 00100012. Hence, the 2’rs complement = 00100012 + 1 = 00100002 = 3410. Finally, the result of the subtraction = -00100002 = -3410.

Next consider 212\(_{10}\)-178\(_{10}\) in binary using addition-based subtraction. However, 212\(_{10}\) = 11010100. But 178\(_{10}\) = 10110010. For -178\(_{10}\), take the 8-bit 2’rs complement of 178. Now, 8-bit 1’s complement of: 178\(_{10}\) = 10110010 is, 01001101. So, the 2’rs complement is, 01001101 + 1 = 01001110. Next add 11010100 to 01001110. So, 11010100 + 01001110 = 100100010. Discarding the 9th bit generated carry yields with 010000012. Here, the MSB is 0. So the result is positive. Hence, 212\(_{10}\)-178\(_{10}\) = 001000012 = 3410.
1) Express the decimal number to be subtracted using the decimal number system as a 8-digit decimal number. Add 0s to the left as required to make it a 8-digit decimal number.

2) Find out the 9’s complement of the decimal number to be subtracted by subtracting each decimal digit from 9.

3) Next, find out the 10’s complement of the decimal number by adding 1 to the Least Significant Digit (LSD) of the 9’s complement obtained in the step above.

4) Express the decimal number from which to subtract as a 8-digit decimal number. If required, add 0’s to the left of the decimal number to make it exactly 8 digits long.

5) Perform addition between the two 8-digit decimal numbers.

6) If there is a generated carry to the 9th digit, simply ignore the carry. The math takes care of this spurious, generated carry from the 8th digit to the 9th digit. In this case, the result of the subtraction is positive, given by the left over 8-digit decimal number.

7) Otherwise, if there is a non-zero value (usually, 9) at the Most Significant Digit (MSD) position, or at the left-most digit position, then the result of the subtraction is negative. Therefore, the negative result now is in the 10’s complement form. Take the 10’s complement of the generated result again by following through the steps 1, 2, and 3 as described above. Next add a negative sign in front of the 10’s complement obtained. This provides you with the negative result of the subtraction. Therefore, the 10’s complement of the 10’s complement of a decimal number gives you the original decimal number, which obviously, is the result of the subtraction with a negative sign (−) in front of it.

Following example illustrates the process.

Example 12 (Base 10’s Complement Example):
Using plain arithmetic, $178_{10} - 212_{10} = -34_{10}$, and $212_{10} - 178_{10} = 34_{10}$. This is shown using the addition-based subtraction approach, as discussed in this paper. Now, $178_{10} = 00000178_{10}$, and $212_{10} = 00000212_{10}$. For $-212_{10}$, take the 10’s complement in 8-digit, as $n = 8$ throughout this paper. The 9’s complement of 00000212$_{10}$ = 99999787$_{10}$. Therefore, the 10’s complement = 99999787$_{10} + 1_{10}$ = 99999822$_{10}$. Next (000000178$_{10}$ + 99999822$_{10}$) = 9999999666$_{10}$. As the left-most 8th digit (MSD) is non-zero, so the result of the subtraction is negative. Take the 10’s complement of 9999999666$_{10}$ again. The 9’s complement of 9999999666$_{10}$ = 00000034$_{10}$. Therefore, the 10’s complement in 8-digit = 00000034$_{10} + 1_{10}$ = 00000035$_{10}$. Removing or adding 0s to the left of a number does not change its numeric value. Hence, 00000034$_{10} = 34_{10}$. So, the actual result of the subtraction = $-34_{10}$.

Next consider $212_{10} + (-178_{10})$. Now, $212_{10} = 00000212_{10}$, and $178_{10} = 00000178_{10}$. The 9’s complement of $00000178_{10} = 99999821_{10}$. So, the 10’s complement in 8-digit = 99999821$_{10} + 1_{10}$ = 99999822$_{10}$. Therefore, $00000212_{10} + (-00000178_{10}) = 00000212_{10} + 99999822_{10}$ = 100000034$_{10}$. Discard the 9th digit generated carry. Hence, the result of the subtraction = 00000034$_{10}$. Since the leftmost 8th digit (MSD) is 0, the result of the subtraction is positive. Hence, the computed result becomes 00000034$_{10} = 34_{10}$. Hence again, the result represents the correct answer.

6. Base 8’s Complement Approach

For the Octal Number System used in computer arithmetic, the base is 8, as there are only 8 unique digits available to represent any number. The digits are 0 through 7. At first, the algorithmic steps for the base’s complement approach are described in the following.

1) Express the decimal number to be subtracted as a 8-digit octal number. Add 0s to the left as required to make it a 8-digit octal number.

2) Find out the 7’s complement of the octal number to be subtracted by subtracting each octal digit from 7.

3) Next, find out the 8’s complement of the octal number by adding 1 to the Least Significant Digit (LSD) of the 7’s complement obtained in the previous step.

4) Express the octal number from which to subtract as a 8-digit octal number. Add 0’s to the left of the octal number as required.

5) Perform addition between the two 8-digit octal numbers.

6) If there is a carry to the 9th digit, simply ignore the generated carry. In this case, the result of the subtraction is positive, given by the remaining 8-digits.

7) Otherwise, if there is a non-zero value (usually, 7) at the Most Significant Digit (MSD) position, or at the left-most digit position, the result of the subtraction is negative. Therefore, the negative result now is in the 8’s complement format. Take again the 8’s complement of the generated result by following through the above steps of 1, 2, and 3, and adding a negative sign in front of the 8’s complement obtained, which gives you the negative result of the subtraction. Therefore, the 8’s complement of the 8’s complement of an octal number gives you the original octal number, which obviously, is the result of the subtraction with a negative sign (−) in front of it.

Following example illustrates the above algorithmic steps.

Example 13 (Octal Base 8’s Complement Example):
Here, $178_{10} = 00000262_{8}$, and $212_{10} = 00000324_{8}$. First, calculate $178_{10} - 212_{10} = 178_{10} + (-212_{10}) = 00000262_{8} + (-00000324_{8})$. Now, for $-00000324_{8}$, take the 8’s complement of 00000324$_{8}$ in 8-digit. So, the number of digits that we are dealing here is, $n = 8$. The 7’s complement of 00000324$_{8} = 77777453_{8}$. The 8’s complement is, 77777453$_{8} + 1_{8} = 77777454_{8}$.
Next \((00000262_8 \pm 7777745_8) = 7777736_8\). As the left-most 8th digit (MSD) is non-zero, so the result of the subtraction is negative. Take the 8’s complement of \(7777736_8\) again. The 7’s complement of \(7777736_8 = 00000041_8\). Therefore, the 8’s complement in 8-digit = \(00000041_8 + 1_8 = 00000042_8\). Removing or adding 0’s to the left of a number does not change its numeric value for any number system. Hence, \(00000042_8 = 42_8\). So, the actual result of the subtraction = \(-42_8 = -34_{10}\).

Next consider \(212_{10} (+(-78_{10}))\) in octal. However, \(212_{10} = 00000324_8\), and \(78_{10} = 00000262_8\). The 7’s complement of \(00000262_8 = 7777751_8\). So, the 8’s complement in 8-digit = \(7777751_8 + 1_8 = 777775168\). Therefore, \(00000324_8 + (-00000262_8) = 00000324_8 + 777775168_8 = 100000042_8\). Discard the the 9th digit generated carry. Hence, the result of the subtraction is, \(00000042_8\). Since the leftmost 8th digit (MSD) is 0, the result of the subtraction is positive. Therefore, the computed result is, \(00000042_8 = 34_{10}\). Obviously, the result obtained represents the correct answer.

7. Complement Approach in Base 16

In literature, the base 16 is represented by the suffix \(H\), which stands for the Hexadecimal Number System (hex for short). The 16 digits, based upon which, the hex number system is built are \(0\) through \(9\), \(A\), \(B\), \(C\), \(D\), \(E\), and \(F\). Following are the algorithmic steps for the addition-based subtraction approach in hexadecimal or in base-16 number system.

1) Express the decimal number to be subtracted as a 8-digit hexadecimal number. Add 0’s to the left of the number as required to make it a perfect 8-digit hexadecimal number.

2) Find out the \(F’s\) complement of the hexadecimal number to be subtracted by subtracting each hex digit from \(F\) (15 in decimal).

3) Next, find out the base 16’s complement of the hexadecimal number by adding 1 to the Least Significant Digit (LSD) of the \(F’s\) complement obtained in the previous step.

4) Express the hexadecimal number from which to subtract as a 8-digit hex number. Add 0’s to the left of the number as required.

5) Perform addition between the two 8-digit hexadecimal numbers.

6) If there is a generated carry to the 9th digit position, simply ignore the carry. The math takes care of this. In this case, the result of the subtraction is positive, given by the remaining 8-digit hexadecimal number.

7) Otherwise, if there is a non-zero value (usually, \(F\)) at the Most Significant Digit (MSD) position, or at the left-most digit position, the result of the subtraction is negative. Therefore, the negative result now is in the 16’s complement format. Take again the 16’s complement of the generated result by following through the aforementioned steps 1, 2, and 3, and adding a negative sign in front of the 16’s complement obtained, which provides the negative result of subtraction. Therefore, the 16’s complement of the 16’s complement of a hexadecimal number provides with the original hex number, which obviously, is the result of the subtraction with a negative sign (\(-\)) in front of it.

Following example clarifies the above algorithmic steps.

Example 14 (Base 16’s Complement Example):
Using the number system conversion, \(178_{16} = 000000B2_{16}\), and \(212_{10} = 000000D4_{16}\). First, compute \(178_{16} - 212_{10} = B2_{16} + (-D4_{16}) = 00000B2_{16} + (-00000D4_{16})\). Now, for \(-00000D4_{16}\), take the base 16’s complement of \(00000D4_{16}\) in 8-digit. So, the number of digits that are being dealt with is, \(n = 8\). The \(F’s\) complement of \(00000D4_{16} = FFFFFFF2B_{16}\). The base 16’s complement is, \(FFFFFFF2B_{16} + 1_{16} = FFFFFF2C_{16}\). Next \((00000B2_{16} + FFFFFF2C_{16}) = FFFFFFDE_{16}\). As the left-most 8th digit (MSD) is non-zero, so the result of the subtraction is negative. Take the base 16’s complement of \(FFFFFDE_{16}\) again. The \(F’s\) complement of \(FFFFFDE_{16}\) is, \(00000216\). Therefore, the base 16’s complement in 8-digit = \(00000021_{16} + 1_{16} = 00000022_{16}\). Removing or adding 0s to the left of a number does not change its numeric value. Hence, \(00000022_{16} = 22_{16}\). So, the actual result of the subtraction = \(-22_{16} = -34_{10}\).

Next consider \(212_{10} (+(-78_{10}))\) in hexadecimal. However, \(212_{10} = 00000D4_{16}\), and \(78_{10} = 00000B2_{16}\). The \(F’s\) complement of \(00000B2_{16} = FFFFFFF4D_{16}\). So, the base 16’s complement in 8-digit = \(FFFFFFF4D_{16} + 1_{16} = FFFFFF4E_{16}\). Therefore, \(00000D4_{16} + (-00000B2_{16}) = 00000D4_{16} + FFFFFF4E_{16} = 10000022_{16}\). Discard the the 9th digit generated carry. Hence, the result of the subtraction is, \(00000022_{16}\). Since the leftmost 8th digit (MSD) is 0, the result of the subtraction is positive. The computed result is given by, \(00000022_{16} = 34_{10}\), which represents the correct solution.

8. Conclusion

Computer CPU utilizes the base’s complement, and the addition-based subtraction in binary number system. The Arithmetic and Logic Unit (ALU) inside the CPU performs the addition-based subtraction using the 2’s Complement of a negative number, as this operation is much faster than the actual subtraction, and also involves less expensive, and optimized hardware circuitry. The control signals for the Op Code comes from the Control Unit to the ALU. As the computing and hardware technology is evolving over the years, so does the related research. This also incorporates technologies involving the base’s complement approach to perform hardware arithmetic.

Representation of positive numbers does not pose an issue in main memory or RAM. However, efficient, and effective representation of negative numbers is a major
research arena in computing. In [3], three representations of negative numbers in computer’s memory are being discussed. These are Sign and Magnitude Representation, the Highest Bit/Digit Complement Representation, and the Base’s Complement Representation. The first two representations are associated with limitations. However, all these limitations are overcome in the Base’s Complement approach, which turns out to be the best for negative numbers. With the Sign and Magnitude Representation, there are two short comings. First of all, +0 and −0 have two different values. Secondly, simple addition and subtraction are required to be performed using different algorithms for positive and negative numbers. The Highest Bit/Digit Complement Representation removes the second problem from the Sign and Magnitude Representation. However, it does not remove the −0 problem. The final, comprehensive solution is obtained through the Base’s Complement Representation, which removes both the problems associated with the Sign and Magnitude Representation. Besides, the hardware circuitry for Base’s Complement Representation is rather easier to design and implement.

For the sign-magnitude representation in base, b, the general case may be expressed as follows in any one of the four computer number systems. Therefore, \( N = \sum_{j=0}^{(n-2)} b^j a_j \), if the number is positive. Also, \( N = -\sum_{j=0}^{(n-2)} b^j a_j \), if the number is negative. A special case with base, b = 2 for the above is represented in [11]. As is easily realizable, sign-magnitude representation is slightly inconvenient as it is using two different representations for positive and negative integers, and renders a little bit of difficulty in testing for 0. On the other-hand, computer CPU design needs to be extremely fast and reliable. Hence, the Arithmetic and Logic Unit (ALU) inside the CPU uses the more advanced base’s complement-based representation. As computer is a binary machine, therefore, it uses the 2’s complement representation for representing the positive and negative integers to be processed by the ALU inside the CPU.

In future, the base’s complement approach for the four computer number systems presented in this paper will be extended to multiplication and division techniques as well. As an example, consider the base’s complement-based multiplication in hexadecimal number system. Here, for simplicity, the data size considered for the multiplicand, and the multiplier is \( n = 2 \). The multiplicand is, \( 10_{10} = A_{16} = 000A_{16} \) (extended to \( 2n = 4 \) digits). The multiplier is \(-7_{10} = FFF9_{16}\) (in base, 16’s complement). Performing multiplication in hex, provides us, \( 000A_{16} \times FFF9_{16} = 9FFB_{16} \). Considering only the rightmost \( 2n = 4 \) digits, and discarding the rest of the leftmost digits, the actual result of the multiplication = \( FFBA_{16} \). Here, the most significant digit (MSD) is \( F \) (the highest digit in hex). So, the result of the multiplication is negative. Taking, base 16’s complement again, the final result of the multiplication = \(-0046_{16} = -70_{10} \), which is correct, and as expected. The direct result of the multiplication between \( 10_{10} \) and \(-7_{10}\) is \( 10_{10} \times \pm 7_{10} = -70_{10} \) in decimal or in base 10 number system. The same result may be obtained using the base 10’s complement approach with \( n = 2 \) digits. Therefore, \( 2n = 4 \). Hence, \( 10_{10} = 0010_{10} \). Also, \(-7_{10} = 9993_{10} \), expressed as the base 10’s complement. Multiplying, \( 0010_{10} \times 9993_{10} = 99930_{10} \). Considering only the rightmost 4 digits, the result of the multiplication is \( 9930_{10} \). Here, the most significant digit is 9, which is the highest digit in decimal number system. Therefore, the result of the multiplication is negative. Taking the base, 10’s complement again, the result of the multiplication = \(-0070_{10} = -70_{10} \), as expected.

Computer hardware uses the base 2’s complement approach in binary multiplication. It uses a faster technique called the Booth’s approach, as computer operations are time critical. The base’s complement approach can be extended to the division process as well. The negative numbers will required to be presented in the base’s complement form in one of the four computer number systems. This remains as a major research in the future.

The Base’s Complement Approach is of paramount importance in computer hardware design. Modern digital computers perform integer multiplications, and integer divisions using the base 2’s complement approach [11]. For checking with the overflow or the underflow error in ALU using the base’s complement approach, the computer uses the Overflow Rule with 2’s complement. Whenever two numbers are added with the same sign, which is either positive or negative, and if the generated result has the opposite sign, which is either negative or positive, then an overflow error has occurred [11]. The same rule applies to the complement-based addition for subtraction in other number systems as well. This paper has significance to the researchers in computer hardware, and hardware optimization areas of interest.

References
