what is the significance of complement operations in binary number system

The most significant role of complement operations in the binary number system, particularly **two's complement**, is to **represent signed (positive and negative) integers** and **simplify arithmetic operations** in digital computer hardware.

This is critical for two main reasons:

1. Representation of Negative Numbers

- **Two's Complement** is the standard method used in modern computers to represent negative numbers. It allows for an efficient and unambiguous representation of the entire range of signed integers.
- The **Most Significant Bit (MSB)**, or leftmost bit, acts as a **sign bit**:
    - **0** indicates a **positive** number.
    - **1** indicates a **negative** number.
- Taking the two's complement of a positive number yields its negative equivalent, and vice versa.

---

2. Simplification of Subtraction

- The core advantage is that it allows the computer's Arithmetic Logic Unit (ALU) to perform **subtraction using the same logic and circuitry as addition**.
- The rule is: **A - B** is calculated as **A + (-B)**, where **-B** is the two's complement of **B**.
- This eliminates the need for a separate, complex subtraction circuit, saving hardware complexity, cost, and improving speed.

why the name 1's and 2's complement

The names **1's complement** and **2's complement** for binary numbers come from the general mathematical concept of "complements" used in number systems to simplify subtraction, particularly in computing.

They are named after the two main types of complements in any number system with a radix (base) of r:

1. Radix Complement (2's Complement in Binary)

- This is the standard, most useful complement for arithmetic.
- It's named after the **radix** (base) of the number system. Since binary is **base 2** (r=2), its radix complement is the **2's complement**.
- **Definition:** For an N-bit number, its 2's complement is defined as the value needed to reach the next power of r=2, which is 2^N.

    - 2^N-Number
- **In Practice:** You find the 1's complement and then **add one** (+1) to it.

Diminished Radix Complement (1's Complement in Binary)

- This is named after the value one less than the radix, which is (radix−1). Since binary is base 2, r−1=2−1=1.
- This is why it's called the **1's complement**.
- **Definition:** For an N-bit number, its 1's complement is the value needed to reach 2^N-1, which is an all-ones number in binary.
    - $(2^N-1)-$Number
- **In Practice:** You simply **invert** (flip) all the bits (0s become 1s, and 1s become 0s).

- The weight representation of a 2's complement binary number is the same as a standard unsigned binary number, *except* for the **Most Significant Bit (MSB)**, which is assigned a **negative weight**.

Bit Weight Formula

For an N-bit 2's complement number, where the bits are indexed from i=0 (Least Significant Bit, LSB) to i=N−1 (Most Significant Bit, MSB), the value W is calculated as follows:

$$W = -(b_{N-1} \cdot 2^{N-1}) + \sum_{i=0}^{N-2} (b_i \cdot 2^i)$$

We need to represent both positive and negative numbers. Three systems are used for representing such numbers:
• Sign-and-magnitude
• 1’s-complement
• 2’s-complement
In all three systems, the leftmost bit is 0 for positive numbers and 1 for negative numbers. Figure 1.3 illustrates all three representations using 4-bit numbers. Positive values have identical representations in all systems, but negative values have different representations. In the sign-and-magnitude system, negative values are represented by changing the most significant bit (b3 in Figure 1.3) from 0 to 1 in the B vector of the corresponding positive value. For example, +5 is represented by 0101, and −5 is represented by 1101. In 1’s-complement representation, negative values are obtained by complementing each bit of the corresponding positive number. Thus, the representation for −3 is obtained by complementing each bit in the vector 0011 to yield 1100. The same operation, bit complementing, is done to convert a negative number to the corresponding positive value.
Converting either way is referred to as forming the 1’s-complement of a given number. For n-bit numbers, this operation is equivalent to subtracting the number from 2n − 1. In the case of the

4-bit numbers in Figure 1.3, we subtract from 24 − 1 = 15, or 1111 in binary. Finally, in the 2's-complement system, forming the 2's-complement of an n-bit number is done by subtracting the number from 2n. Hence, the 2's-complement of a number is obtained by adding 1 to the 1's-complement of that number.

Note that there are distinct representations for +0 and −0 in both the sign-and magnitude and 1's-complement systems, but the 2's-complement system has only one representation for 0. For 4-bit numbers, as shown in Figure 1.3, the value −8 is representable in the 2's-complement system but not in the other systems. The sign-and-magnitude system seems the most natural, because we deal with sign-and-magnitude decimal values in manual computations. The 1's-complement system is easily related to this system, but the 2's-complement system may appear somewhat unnatural. However, we will show that the 2's-complement system leads to the most efficient way to carry out addition and subtraction operations. It is the one most often used in modern computers.

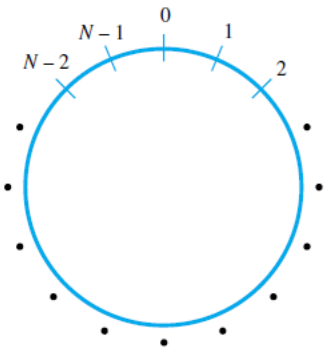
B	Values represented		
$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

Addition and Subtraction of Signed Integers

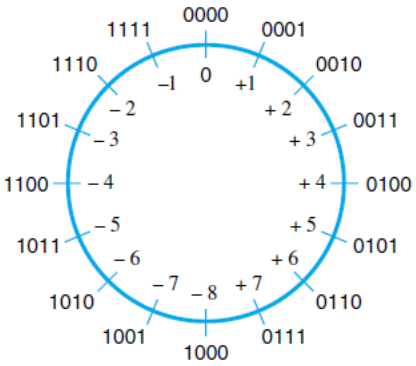
We introduced three systems for representing positive and negative numbers, or, simply, signed numbers. These systems differ only in the way they represent negative values. Their relative merits from the standpoint of ease of performing arithmetic operations can be summarized as follows. The sign-and-magnitude system is the simplest representation, but it is also the most awkward for addition and subtraction operations. The 1's-complement method is somewhat better. The 2's-complement system is the most efficient method for performing addition and subtraction operations.

To understand 2's-complement arithmetic, consider addition modulo N (abbreviated as mod N). A helpful graphical device for the description of addition of unsigned integers mod N is a circle

with the values 0 through N − 1 marked along its perimeter, as shown in Figure 1.5a. Consider the case N = 16, shown in part (b) of the figure. The decimal values 0 through 15 are represented by their 4-bit binary values 0000 through 1111 around the outside of the circle. In terms of decimal values, the operation (7 + 5) mod 16 yields the value 12. To perform this operation graphically, locate 7 (0111) on the outside of the circle and then move 5 units in the clockwise direction to arrive at the answer 12 (1100). Similarly, (9 + 14) mod 16 = 7; this is modeled on the circle by locating 9 (1001) and moving 14 units in the clockwise direction past the zero position to arrive at the answer 7 (0111). This graphical technique works for the computation of (a + b) mod 16 for any
unsigned integers a and b; that is, to perform addition, locate a and move b units in the clockwise direction to arrive at (a + b) mod 16.



(a) Circle representation of integers mod *N*



(b) Mod 16 system for 2's-complement numbers

Figure 1.5 Modular number systems and the 2's-complement system.

Now consider a different interpretation of the mod 16 circle. We will reinterpret the binary vectors outside the circle to represent the signed integers from −8 through +7 in the 2's-complement representation as shown inside the circle. Let us apply the mod 16 addition technique to the example of adding +7 to −3. The 2's-complement representation for these numbers is 0111 and 1101, respectively. To add these numbers, locate 0111 on the circle in Figure 1.5b. Then move 1101 (13) steps in the clockwise direction to arrive at 0100, which yields the correct answer of +4. Note that the 2's-complement representation of −3 is interpreted as an unsigned value for the number of steps to move.
If we perform this addition by adding bit pairs from right to left, we obtain

```
   0 1 1 1
+  1 1 0 1
-------------
1  0 1 0 0
↑
Carry-out
```

If we ignore the carry-out from the fourth bit position in this addition, we obtain the correct answer. In fact, this is always the case. Ignoring this carry-out is a natural result of using mod N arithmetic. As we move around the circle in Figure 1.5b, the value next to 1111 would normally be 10000. Instead, we go back to the value 0000. The rules governing addition and subtraction of n-bit signed numbers using the 2'scomplement representation system may be stated as follows:

• To ***add*** two numbers, add their n-bit representations, ignoring the carry-out bit from the most significant bit (MSB) position. The sum will be the algebraically correct value in 2's-complement representation if the actual result is in the range-2^{n-1} through$+2^{n-1} - 1$.

• To ***subtract*** two numbers X and Y , that is, to perform X − Y , form the 2's-complement of Y , then add it to X using the add rule. Again, the result will be the algebraically correct value in 2's-complement representation if the actual result is in the range -2^{n-1} through $+2^{n-1} - 1$.

Figure 1.6 shows some examples of addition and subtraction in the 2's-complement system. In all of these 4-bit examples, the answers fall within the representable range of −8 through +7. When answers do not fall within the representable range, we say that arithmetic overflow has occurred. A later subsection discusses such situations. The four addition operations (a) through (d) in Figure 1.6 follow the add rule, and the six subtraction operations (e) through (j) follow the subtract rule. The subtraction operation requires forming the 2's-complement of the subtrahend (the bottom value). This operation is done in exactly the same manner for both positive and negative numbers.

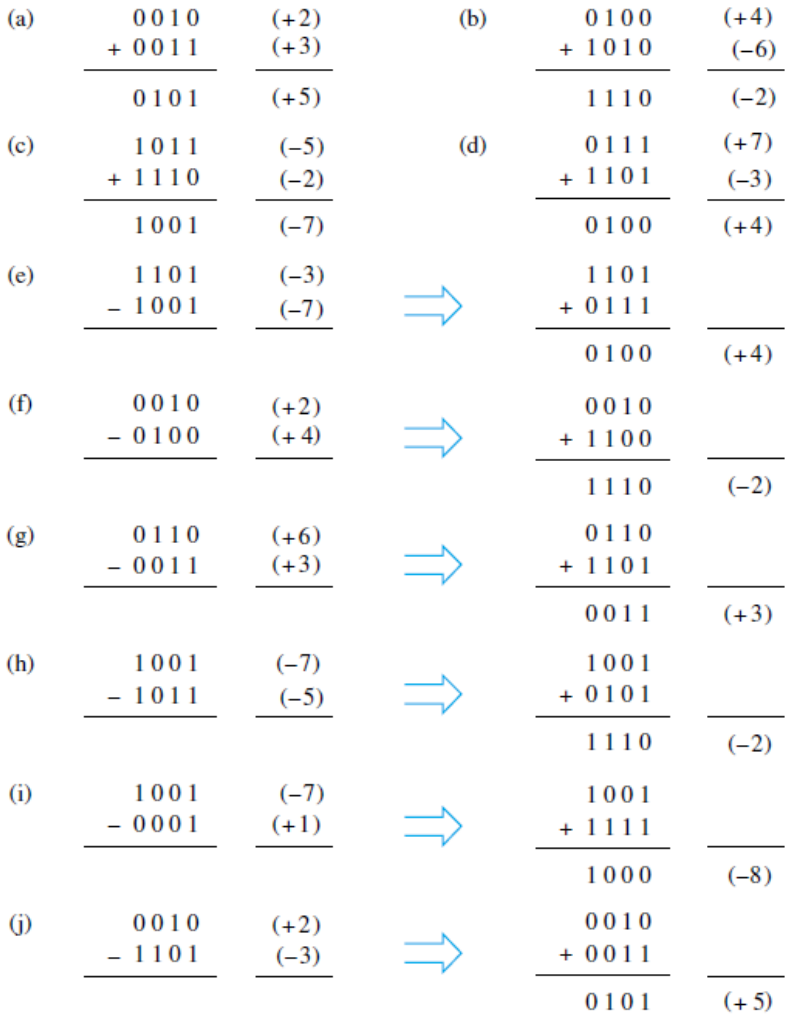


Figure 1.6 2's-complement Add and Subtract operations.

To form the 2's-complement of a number, form the bit complement of the number and add 1. The simplicity of adding and subtracting signed numbers in 2's-complement representation is the reason why this number representation is used in modern computers. It might seem that the 1's-complement representation would be just as good as the 2's-complement system. However, although complementation is easy, the result obtained after an addition operation is not always correct. The carry-out, cn, cannot be ignored. If cn = 0, the result obtained is correct. If cn = 1, then a 1 must be added to the result to make it correct. The need for this correction operation means that addition and subtraction cannot be implemented as conveniently in the 1's-complement system as in the 2's-complement system.

Sign Extension

We often need to represent a value given in a certain number of bits by using a larger number of bits. For a positive number, this is achieved by adding 0s to the left. For a negative number in 2's-complement representation, the leftmost bit, which indicates the sign of the number, is a 1. A longer number with the same value is obtained by replicating the sign bit to the left as many times as needed. To see why this is correct, examine the mod 16 circle of Figure 1.5b. Compare it to larger circles for the mod 32 or mod 64 cases. The representations for the values −1, −2, etc.,

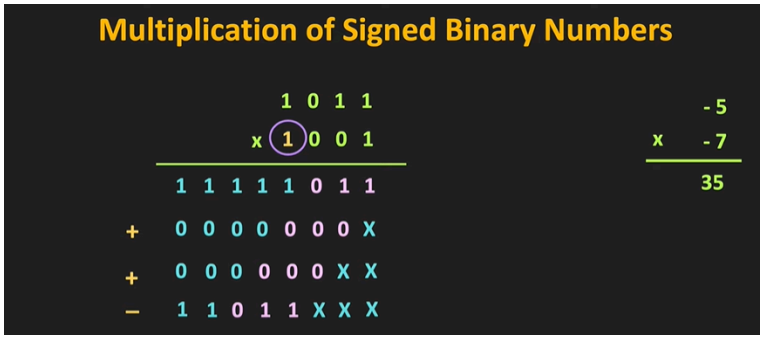
are exactly the same, with 1s added to the left. In summary, to represent a signed number in 2's-complement form using a larger number of bits, repeat the sign bit as many times as needed to the left. This operation is called sign extension.

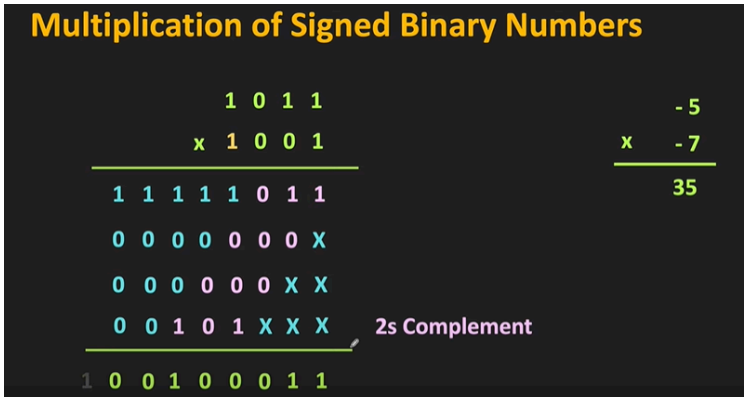
Overflow in Integer Arithmetic

Using 2's-complement representation, n bits can represent values in the range -2^{n-1} to $+2^{n-1} - 1$. For example, the range of numbers that can be represented by 4 bits is −8 through +7, as shown in Figure 1.3. When the actual result of an arithmetic operation is outside the representable range, an arithmetic overflow has occurred.

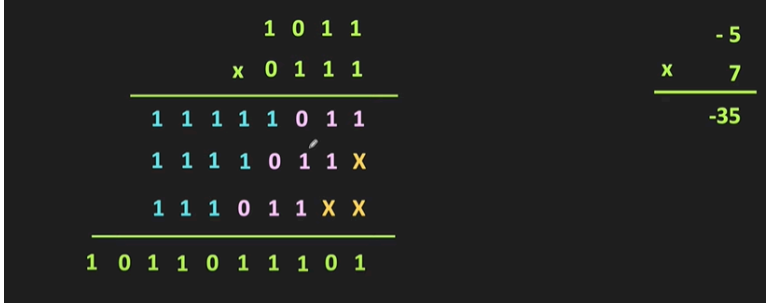
When adding unsigned numbers, a carry-out of 1 from the most significant bit position indicates that an overflow has occurred. However, this is not always true when adding signed numbers. For example, using 2's-complement representation for 4-bit signed numbers, if we add +7 and +4, the sum vector is 1011, which is the representation for −5, an incorrect result. In this case, the carry-out bit from the MSB position is 0. If we add −4 and −6, we get 0110 = +6, also an incorrect result. In this case, the carry-out bit is 1. Hence, the value of the carry-out bit from the sign-bit position is not an indicator of overflow**. Clearly, overflow may occur only if both summands have the same sign. The addition of numbers with different signs cannot cause overflow because the result is always within the representable range.**

These observations lead to the following way to detect overflow when adding two numbers in 2's-complement representation. Examine the signs of the two summands and the sign of the result. When both summands have the same sign, an overflow has occurred when the sign of the sum is not the same as the signs of the summands.





Multiplication of Signed Binary Numbers



Multiplication of Signed Binary Numbers