Binary number system

A **number system** is defined by the number of digits used for calculations.
•The **decimal number system**, which we use in everyday life and maths, uses 10 digits (0-9). This is also called **Base 10**.
•The **binary number system** uses only two digits (0 and 1). This is called **Base 2**.
•Other number systems exist, such as **Hexadecimal** (Base 16) and **Octal** (Base 8). The video focuses on the binary number system in detail.

In the binary number system, all numbers consist only of 0s and 1s. For example, 101010 is a binary number, whereas 122396 is a decimal number. The video explains how to **convert numbers between the decimal and binary systems**.

Converting Decimal Numbers to Binary Numbers
This process involves **repeated division of the decimal number by 2**.
•You repeatedly divide the number by 2 and note the **remainder** (either 0 or 1) on the right side.
•You continue dividing the quotient by 2 until the quotient becomes 0.
•The binary form is obtained by reading the remainders from **bottom to top** (backwards).

•**Example (converting 42):**
◦42 / 2 = 21, Remainder = 0
◦21 / 2 = 10, Remainder = 1
◦10 / 2 = 5, Remainder = 0
◦5 / 2 = 2, Remainder = 1
◦2 / 2 = 1, Remainder = 0
◦1 / 2 = 0, Remainder = 1
◦Reading from bottom to top: **101010**. Thus, 42 in base 10 is equivalent to 101010 in base 2.

•**Example (converting 50):**
◦50 / 2 = 25, Remainder = 0
◦25 / 2 = 12, Remainder = 1
◦12 / 2 = 6, Remainder = 0
◦6 / 2 = 3, Remainder = 0
◦3 / 2 = 1, Remainder = 1
◦1 / 2 = 0, Remainder = 1

◦Reading from bottom to top: **110010**. This is the binary form of 50.

•**Coding logic for decimal to binary conversion:**
•Use a loop that continues as long as the decimal number is greater than 0.
•Inside the loop:
◦Calculate the **remainder** by taking the decimal number modulo 2 (`decimal_number % 2`).
◦Update the decimal number by dividing it by 2 (`decimal_number = decimal_number/2`).

◦To build the binary number (which is read backwards in the manual method), the code constructs it by multiplying the remainder by increasing powers of 10 and adding it to an `answer` variable.
◦A `power` variable, initialised to 1 (representing 10^0), is used.
◦The remainder is multiplied by the current `power` and added to the `answer`.
◦The `power` is then updated by multiplying it by 10 for the next digit (`power = power * 10`). This effectively places the next remainder in the correct decimal place value (tens, hundreds, etc.) to represent its position in the binary number.

•The final `answer` variable holds the binary representation as a decimal integer (e.g., 101010 is stored as the integer 101010).
•A dry run with the number 6 is shown to demonstrate the code logic.
•C++ code for the `convertDecimalToBinary` function is provided and demonstrated, showing the conversion of 50 to 110010 and printing binary forms for numbers 1 to 10.

Converting Binary Numbers to Decimal Numbers
To convert a binary number back to its decimal form, you multiply each digit by powers of 2 and sum the results.
•Start from the **rightmost digit** (the least significant bit).
•Multiply the rightmost digit by 2 to the power of 0 (2^0 = 1).
•Move left, multiply the next digit by 2 to the power of 1 (2^1 = 2).
•Continue moving left, increasing the power of 2 for each digit (2^2, 2^3, etc.).
•**Sum** all the results to get the final decimal number.

•**Example (converting 101010):**
◦Starting from the right:
◦0 * 2^0 = 0 * 1 = 0
◦1 * 2^1 = 1 * 2 = 2
◦0 * 2^2 = 0 * 4 = 0
◦1 * 2^3 = 1 * 8 = 8
◦0 * 2^4 = 0 * 16 = 0
◦1 * 2^5 = 1 * 32 = 32
◦Sum: 0 + 2 + 0 + 8 + 0 + 32 = **42**. This matches the previous example.

•**Example (converting 110010):**
◦Starting from the right:
◦0 * 2^0 = 0
◦1 * 2^1 = 2
◦0 * 2^2 = 0
◦0 * 2^3 = 0
◦1 * 2^4 = 16
◦1 * 2^5 = 32
◦Sum: 0 + 2 + 0 + 0 + 16 + 32 = **50**. This also matches a previous example.

•**Coding logic for binary to decimal conversion.**
•Use a loop that continues as long as the binary number (represented as a decimal integer) is greater than 0.
•Inside the loop:
◦Get the **last digit** of the binary number by taking it modulo 10 (`binary_number % 10`). This is the remainder when dividing by 10.
◦Multiply this last digit (remainder) by the current **power of 2** (`power`)8. Add this result to an `answer` variable.
◦Update the binary number by integer division by 10 (`binary_number = binary_number / 10`) to remove the last digit.
◦Update the `power` variable by multiplying it by 2 (`power = power * 2`) for the next digit. The `power` variable is initialised to 1 (representing 2^0).
•The final `answer` variable holds the decimal form of the number.
•C++ code for the `binaryToDecimal` function is provided and demonstrated, showing the conversion of 101 to 5 and 1010 to 10.
Common Binary Forms (0-10)
•0: 0
•1: 1
•2: 10
•3: 11
•4: 100
•5: 101
•6: 110
•7: 111
•8: 1000
•9: 1001
•10: 1010
Note that adding leading zeros to a binary number does not change its value (e.g., 001 is the same as 1).

Alternative Decimal to Binary Conversion Trick: Instead of repeated division, you can use the positional values (powers of 2) to convert a decimal number to binary.
•List the powers of 2: 1, 2, 4, 8, 16, 32, 64, 128, ... These represent the values contributed by each bit position (from right to left: 2^0, 2^1, 2^2, etc.).
•To find the binary form of a number (e.g., 25), find the largest power of 2 that is less than or equal to the number (e.g., 16 for 25).
•Place a '1' in the position corresponding to that power of 2 and subtract that power from the number (25 - 16 = 9).
•Repeat the process with the remaining number (9) and the remaining powers of 2. Find the largest power of 2 less than or equal to 9 (which is 8). Place a '1' in the '8' position and subtract (9 - 8 = 1)
•Repeat with the remaining number (1) and powers of 2. The largest power of 2 less than or equal to 1 is 1 (2^0). Place a '1' in the '1' position and subtract (1 - 1 = 0).
•For any power of 2 positions not used in the sum, place a '0'.
•So, for 25: 16 (1), 8 (1), 4 (0), 2 (0), 1 (1) -> **11001**.

•**Example (converting 36):** Powers of 2: ..., 32, 16, 8, 4, 2, 1.
◦32 is <= 36. Place '1' at 32 position. Remaining: 36 - 32 = 4.
◦16 is > 4. Place '0' at 16 position.

◦8 is > 4. Place '0' at 8 position.
◦4 is <= 4. Place '1' at 4 position. Remaining: 4 - 4 = 0.
◦2 is > 0. Place '0' at 2 position.
◦1 is > 0. Place '0' at 1 position.
◦Binary form of 36: ..., 32 (1), 16 (0), 8 (0), 4 (1), 2 (0), 1 (0) -> **100100**.
Insight on Odd Numbers in Binary When converting to decimal, all powers of 2 except 2^0 (which is 1) contribute an even value. Therefore, for a binary number to represent an **odd decimal number**, the rightmost bit (the 2^0 position) **must be 1**.

Binary Number Addition Addition in binary follows these rules:
•0 + 0 = 0
•0 + 1 = 1
•1 + 0 = 1
•**1 + 1 = 10** (which is 0 with a **carry** of 1 to the next left position).

Storing Numbers in Computer Memory
Numbers are stored in memory using their binary form.
•**Positive Numbers:** For a standard integer (often 4 bytes or 32 bits), the positive binary form is padded with leading zeros to fill the allocated space. For example, the binary form of 10 (1010) stored in 32 bits would be `0000...00001010`.

•**Negative Numbers:** Negative numbers are stored using a special representation called **Two's Complement**.

Two's Complement Two's Complement is calculated for a negative number (e.g., -10) as follows:
1.Find the binary form of the **positive** equivalent number (e.g., 10 is 1010).
2.**Prefix** this binary form with a 0. This leftmost bit is called the **Most Significant Bit (MSB)**. Currently, 0 indicates a positive number. (Example: 01010). *Note: The video seems to imply adding a single 0, but for a fixed bit size like 32 bits, it would typically be padded with 0s to the required length first.*
3.Calculate the **One's Complement** of this prefixed number. One's Complement is obtained by **flipping every bit** (changing 0s to 1s and 1s to 0s). (Example for 01010: 10101).
4.Add **1** to the result of the One's Complement. (Example for 10101: 10101 + 1).
◦10101
    ▪1
◦**10110**(using binary addition: 1+1=0 carry 1, 0+0+carry 1=1, etc.)
5.This final binary form (**10110** in the example) is the **Two's Complement** representation of the negative number (-10. The **MSB** (the leftmost bit) in Two's Complement indicates the sign: **0 for positive**, **1 for negative**.

Converting a Negative Binary Number (Two's Complement) back to Decimal
If you are given a binary number and you know it represents a negative number (because its MSB is 1), you convert it back to decimal by finding its Two's Complement again.
1.Take the given negative binary number (e.g., 10110).
2.Calculate its **One's Complement** by flipping all bits (10110 becomes 01001).
3.Add **1** to the One's Complement result (01001 + 1 = 01010).
4.Convert the resulting binary number (01010) to its decimal form using the standard binary-to-decimal conversion method.
◦01010 = (0 * 2^4) + (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0) = 0 + 8 + 0 + 2 + 0 = 10.

5.Since you knew the original binary number was negative (MSB was 1), the decimal equivalent is the **negative** of this result. Therefore, 10110 represents **-10**.
An example converting -8 to binary (1100) and back to -8 is also demonstrated, following the same Two's Complement process.

Understanding the binary number system, decimal-to-binary and binary-to-decimal conversion, and Two's Complement is **crucial for understanding future topics** like **bitwise operators**, which operate on the binary forms of numbers