Number Systems

There are 4 kinds of number systems a computer can understand and follow:

1. Binary : radix 2 – 0 and 1
2. Octal : radix 8 – 0 to 7
3. Decimal : radix 10 – 0 to 9 (Commonly used by humans)
4. Hexadecimal : radix 16 – 0 to 9 and A to F

Here is a list showing how each of them is written from 0 to 16 of the decimal system:

Decimal	**Binary**	**Octal**	**Hexadecimal**
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

There are two ways to write these number systems. Consider the number 1:

	Decimal	Binary	Octal	Hexadecimal
Human-readable	1 or $(1)_{10}$	$(1)_2$	$(1)_8$	$(1)_{16}$
Computer-readable	1	0b1	0o1	0x1

In general, any number with radix X, having m digits to the left of the decimal point and n digits after the decimal point can be expressed as:

$$a_m(X)^{m-1} + a_{m-1}(X)^{m-2} + \ ... \ + a_2(X)^1 + a_1(X)^0 \ . \ b_1(X)^{-1} + b_2(X)^{m-2} + \ ... \ + b_{n-1}(X)^{n-1} + b_n(X)^{-n}$$

Where a_m is the m^{th} digit on the right and b_n is the n^{th} digit on the left.

Conversions

There is a possible way to rewrite any number of one number system to any of the other three systems *without* using the table.

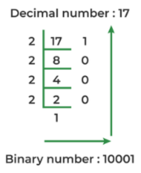
Decimal → Binary, Octal, Hexadecimal

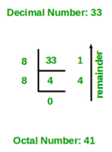
To convert any decimal number to another number system, follow these steps:

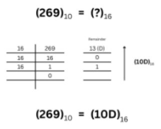
1. Keep dividing the number with the radix of the new system.

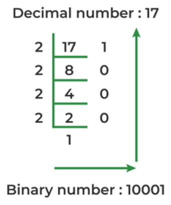
2. Stop when you reach 0
3. Collect the reminders after division, and write them in the reverse order.

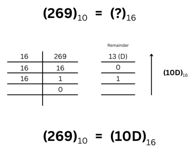
Example:

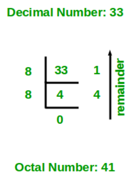












Binary → Decimal, Octal, Hexadecimal

To convert binary numbers to **decimal**, follow these steps:

1. Take all digits that are 1
2. Do $2^{\text{index of the digit from right}}$ in each case
3. Add all of them

Example: Take the number $(110)_2$.

To convert this, we have 1 in index 1 and 2 from the right. So we find 21 and 22, which are 2 and 4 respectively. Now we add the numbers: 2 + 4 = 6. So $(110)_2$ = 6

To convert it into **octal**, and **hexadecimal**:

1. Split the binary number into pairs of 3 (for octal) and 4 (for hexadecimal).
2. If a pair has insufficient digits, add zeros to the left, until there is a proper pair.
3. Convert each binary pair into its corresponding decimal form.
4. If the decimal number is 10 or greater (only in hexadecimal), convert it to A - F.

Octal → Decimal, Binary, Hexadecimal

To convert octal numbers to **decimal**, follow these steps:

1. Take all digits that are 1 - 7.

2. Do $8^{\text{index of the digit from right}}$ in each case, and multiply it with the digit extracted.
3. Add all of them

Example: Take the number $(706)_8$.

To convert this, we have 6 in index 0 and 7 in index 2 from the right. So we find 80 and 82, which are 1 and 64 respectively. Now, multiply it with the digits extracted. We have 1 × 6 and 64 × 7. Finally, we add the numbers: 6 + 448 = 454. So (706)8 = 454

To convert octal numbers to **binary**, just do the reverse process. Write each octal digits binary notation as if they were decimal digits.

To convert octal numbers to **hexadecimal**, first, convert octal to binary, and then the binary notation to hexadecimal.

Example: $(64)_8 = (110100)_2 = (110100)_2 = (00110100)_2 = (34)_{16}$

Hexadecimal → Decimal, Binary, Octal

To convert hexadecimal numbers to **decimal**, follow these steps:

1. Take all digits that are 1 - F.
2. Convert A - F to 10 - 16 (view table)
3. Do 16index of the digit from right in each case, and multiply it with the digit extracted.
4. Add all of them

Example: Take the number $(3D)_{16}$.

To convert this, we have D in index 0 and 3 in index 1 from the right. D gets converted to 13. So we find 160 and 161, which are 1 and 16 respectively. Now, multiply it with the digits extracted. We have 1 × 13 and 16 × 3. Finally, we add the numbers: 13 + 48 = 61. So $(3D)_{16} = 61$

To convert octal numbers to **binary**, just do the reverse process. Write each octal digits binary notation as if they were decimal digits.

To convert octal numbers to **octal**, first, convert hexadecimal to binary, and then the binary notation to octal.

Example: $(BA)_{16} = (1011\ 1010)_2 = (10\ 111\ 010)_2 = (010\ 111\ 010)_2 = (272)_8$

Floating Point Numbers

Consider a decimal number having floating point values. To convert it into a number of radix X, follow these steps:

1. For the integer part of the number, do the normal conversion
2. Extract the integer part of the number and make the integer in the number 0.

1. Multiply the new number with X.
2. Repeat from step 2 for the new result, until the result after step 3 becomes 0
3. Write the extracted integers from top to bottom. This is the number written in the new form.

Example:
Consider the number $(10.6)_{10}$
Now, 10 = $(1010)_2$
We have 0.625:
$0.625 \cdot 2 = 1.25 \to 1$
$0.25 \cdot 2 = 0.5 \to 0$
$0.5 \cdot 2 = 1 \to 1$
Now we have 0 so we stop here. The decimal part is $(101)_2$
The entire number becomes $(1010.101)_2$

Binary IEEE 754 Representation

This binary number is written in the form `Sign ExponentBias Mantissa`.

The sign is of length **1 bit**, the exponent bias of **8 bits** (For a 32-bit float) and the mantissa (the floating binary representation).

For this:

1. First find the binary representation for the number.
2. Write it in exponential form
3. Get the exponent (written as $1.\text{xxxx} \cdot 2^n$), add it to 127 (for 32-bit mantissa) and convert this number to binary
4. Extract the mantissa from the $1.\text{xxxx} \cdot 2^n$
5. Find the sign bit (0 means positive and 1 means negative)
6. Write it all together

> For example, consider $(4.2)_{10}$
> Following the above steps, $4.2 = 100.00110\ldots$
> To write it in exponent form, convert it to $1.$ format. Which means the above number becomes $1.0000110\ldots \cdot 2^2$. And the exponent part is 2.
> According IEEE 754, the "bias" tells us to add 127 for a 32-bit mantissa. So exponent = $2 + 127 = 129.$
> Which becomes 10000001 when converted to binary.
> The mantissa is the float value from the second position after the decimal point extracted from the $1.$ so it becomes $0000110\ldots$
> The sign is positive, so the sign bit is 0.
> So the final number becomes $0\,10000001\,0000110\ldots$

Binary Systems

Along with the original binary notation, there are other variants to represent a number using only 2 digits (or bits). The most commonly used ones are **Gray code** and **BCD**.

Binary Coded Decimal (BCD)

This form of binary encoding, as the name suggests, is basically just writing each decimal digit in its binary form, which is later combined as BCD. Here are the major differences between just binary and BCD:

Binary	Binary Coded Decimal (BCD)
Base 2	Base 10 encoded in binary
Whole number in binary	Each decimal digit is a 4 bit binary
Space wise efficient	Takes a lot of space
Needs math (continuous division by 2)	Direct conversion
Understood by computers, and processors	More easily understood by humans as compared to binary

So for example, lets take the decimal number 13.

Now, in binary 13 is $(1101)_2$ upon using the conversion procedure.

In BCD, we take the digits separately. i.e. 1 and 3, and convert each digit to binary, and then put it back together. So it would be like this:

> 13 → 1 and 3 → 0001 and 0011 → $(0001\,0011)$

Gray Code

Unlike binary, in this form of binary encoding, only one bit is flipped when counting in decimal system.

Decimal	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

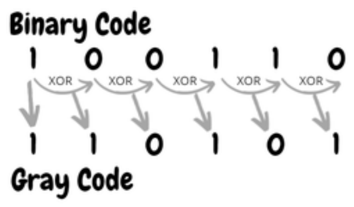
It can be noticed that only one bit flips when moving across numbers.

This encoding is really helpful in places where transition error occurs due to flipping multiple bits at once. It mainly happens in encoders and ADCs (Analogue to Digital Converter).

Binary → Gray Code

To convert a binary number to gray code:

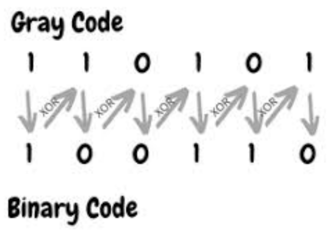
- MSB (Most Significant Bit) remains the same
- The next Gray Code bit is the result of the XOR operation on the current Binary bit and the previous one.



Gray Code → Binary

To convert a gray code number to binary:

- MSB (Most Significant Bit) remains the same.
- The next Binary bit is the result of the XOR of the current Gray Code bit and the previous Binary bit.



Error Checking Parity

While transmitting data, there is always a chance that some unknown noise flips the bits of the data transmitted, eventually misleading it. This is where parity comes into play. It checks if any bit has been flipped in the data recieved, basically detecting an error.

This is a simple, low-cost method to check if an error has occured during transmission of data or storage.

There are two kinds of parities — **odd parity** and **even parity**.

Odd Parity

When the data (in binary) has odd number of 1s, then the parity bit becomes 0 to keep the number of 1s same. If the data has even number of 1s, then the parity bit becomes 1 to change the number of 1s to odd.

Logically, it can be done using the XOR gate. Each bit can passed into the XOR gate, and the result tells what the parity bit is supposed to be.

If a binary data is $b = b_1b_2b_3b_4$, then parity bit is $p = b_1 \oplus b_2 \oplus b_3 \oplus b_4$.

Even Parity

When the data (in binary) has even number of 1s, then the parity bit becomes 0 to keep the number of 1s same. If the data has odd number of 1s, then the parity bit becomes 1 to change the number of 1s to odd.

Logically, it can be done using the XNOR or XAND gate, which is the opposite of XOR gate. Each bit can passed into the XAND gate, and the result tells what the parity bit is supposed to be.

If a binary data is $b = b_1b_2b_3b_4$, then parity bit is $p = b_1 \otimes b_2 \otimes b_3 \otimes b_4$.

Now, this parity bit p is added to the end of the data in binary and transferred. Here is a reference for how parity bit varies and how data looks with the parity bit:

Decimal	Binary	Odd Parity	Even Parity
0	000	1 → 000 1	0 → 000 0
1	001	0 → 001 0	1 → 001 1
2	010	0 → 010 0	1 → 010 1
3	011	1 → 011 1	0 → 011 0
4	100	0 → 100 0	1 → 100 1