Yash Raj ICT Lab 25K-0878

**Introduction**

We all use the decimal system every day. it’s just the numbers from 0 to 9. But have you ever wondered how computers actually "see" numbers? It turns out they use a much simpler, but for us, a much longer system called **binary**, which is just 0s and 1s. Trying to read and write in only 0s and 1s is really tough for people. It's like trying to read a book where every word is spelled out with only two letters!

To make things easier, programmers use two other number systems as helpful shortcuts: **octal** (base-8) and **hexadecimal** (base-16). This report will look at how these different systems are actually used in real computing, from the computer's brain to the programmer's screen.

**Binary**

At the very heart of every computer, everything is binary. The processor, the memory, and the storage all work with billions of tiny switches that are either on (1) or off (0). This is called machine-level programming.

**How it is really used:** Every single command you give to a computer, like adding two numbers or saving a file, gets translated down into a long string of 0s and 1s that the CPU can understand. These are the machine code instructions.

Example: Let’s add 6 and 2.

* In decimal, it's easy: 6 + 2 = 8.
* In the computer's world, it uses binary:
* 6 in binary is 110
* 2 in binary is 010
* The computer adds them: 110 + 010 = 1000 (which is 8 in decimal).

While this is super-efficient for the machine, imagine if a programmer had to find an error in a program that looked like this: 10111001010101101101. It would be almost impossible! This is why we need the other systems.

**Hexadecimal**

The Programmer’s best friend

Hexadecimal, or "hex," is the most common shortcut used by programmers. It's a base-16 system, so it uses the digits 0-9 and then the letters A-F (where A=10, B=11, up to F=15). It's especially useful for working with memory addresses.

**The Link to Memory:** Every byte of your computer's RAM has a unique address so the CPU knows where to find data. These addresses are really just big binary numbers, but they are almost always shown to programmers in hex.

**Why Hex is so Handy:** The main reason is that one hex digit perfectly represents four binary digits (which is called a nibble). This makes converting between binary and hex really easy.

**An Example from Class:** Let's convert a binary number to hex.

Take the binary address: 1101 1010 1110 1101

The first step is to split it into groups of four: 1101 | 1010 | 1110 | 1101

Now, convert each group:

* 1101 = 13 in decimal, which is **D** in hex.
* 1010 = 10 in decimal, which is **A** in hex.
* 1110 = 14 in decimal, which is **E** in hex.
* 1101 = 13 again, which is **D** in hex.

So, the long, confusing binary string becomes the much shorter and clearer hex address: **DAED**.

So, when a programmer is debugging and sees an error at memory location 0x7ffd12a45678, they are looking at a hexadecimal number. This makes their job a lot easier.

**Octal**

Octal is a base-8 system that uses digits from 0 to 7. It was more popular in the old days of computing with systems like the PDP-8. It groups binary digits into sets of three.

**Why it was used:** Older computers often had word sizes of 12, 24, or 36 bits. Since these numbers are divisible by 3, grouping bits into sets of three for octal was a natural fit.

**Where it's still used today:** The most common place we see octal today is in **Unix and Linux file permissions**.

* In Linux, files have permissions for three groups: the owner, the group, and everyone else.
* The permissions are Read, Write, and Execute, which can be thought of as three bits (on or off).
* These three bits are perfectly represented by one octal digit (0-7).

**Example:** The permission code 755 that you might see is in octal.

* The first digit, 7, means 111 in binary, so the owner can Read, Write, and Execute.
* The second digit, 5, means 101 in binary, so the group can Read and Execute, but not Write.

**Comparison and Conclusion**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| System | Base | Digits Used | Main Use | Good for | Bad for |
| Binary | 2 | 0 – 1 | Machine code, CPU instructions | The computer itself; it's the only thing it truly understands. | Humans; it's too long and complicated to work with directly. |
| Octal | 8 | 0 – 7 | Older systems, Unix file permissions | Grouping bits in 3s; representing 3-bit values neatly. | Modern 8-bit based systems; the grouping doesn't fit as nicely as hex. |
| Hex | 16 | 0 – 9, A – F | Memory addresses, debugging, web colors | Grouping bits in 4s; it's a perfect fit for bytes (8 bits = 2 hex digits). | Being intuitive; you have to learn the letters A-F |

**Advantages and Limitations**

1. **Binary** is perfect for the machine but a nightmare for people. Its advantage is its simplicity for hardware, but that's also its biggest limitation for programmers.
2. **Hexadecimal** is the best balance for humans. It's compact and maps directly to binary in a way that fits modern computer architecture (which is all based on bytes). Its only downside is that it looks a little strange at first.
3. **Octal** is a specialist. It was great for its time, but it's not as universally useful as hex anymore. It's still important to know for certain tasks, like managing Linux servers.

**Conclusion**

In the end, all these number systems work together. **Binary** is the unchangeable foundation—the computer will always think in 0s and 1s. **Hexadecimal** is the essential translator that allows programmers to understand what the computer is doing without losing their minds. And **Octal**, while not as common, still has its specific job. Learning about them doesn't just help us convert numbers for class; it helps us understand the very building blocks of the digital world we live in.