**Assignment on Number System Conversion and Data Representation Codes**

In the technology field, especially in companies working with digital systems, the ability to convert data between different number systems can be a desirable or even required skill. The simplest technique consists of dividing a decimal number by the base of the target system. This process is clearly explained by InfoPack in *Number System Conversion Techniques*.

The method works as follows: divide the decimal number by the target base (for example, 2 for binary, 8 for octal, or 16 for hexadecimal), keep the remainder (if no remainder exists, place a 0 instead), and repeat the operation using the integer quotient until the result is zero. Each remainder is always less than the base and represents a "digit" in the new system. Finally, the remainders are read in reverse order: from the **Least Significant Bit (LSB)** to the **Most Significant Bit (MSB)**, which corresponds to the remaining of the first division.

For example, if we take the number 270 (the first three digits of my birth date), converting it to binary results in 100001110, to octal 416, and to hexadecimal 10E. The remainders from each division represent the bits of the number in the new system.

This type of procedure can be automated by creating a function that takes a number and a base as parameters and returns the remainders in the correct order. This is useful in areas such as **data analysis**, where information often needs to be transformed between formats for proper integration or visualization, and also in **electronic engineering** or **embedded systems**, where data is communicated using binary or hexadecimal formats due to hardware compatibility.

Regarding **representation codes**, I learned that not all of them are appropriate for every context. The **ASCII code**, for example, is an efficient standard for representing alphanumeric characters using numeric values, but it only covers 128 characters, which limits it to English and a few symbols. According to Bowne (2021), each ASCII character is represented as an 8-bit byte. For example, the capital letter A is represented by the decimal value 65, which corresponds to the binary 01000001. This means that a word like “ABC” becomes a 24-bit sequence when interpreted in binary (Bowne, S. (2021). *Hands-On Cryptography with Python*).

These same principles are applied in **Base64 encoding**, where sequences of bytes are converted into readable text to enable secure data transmission.

However, if the system must handle multiple languages (for example, in a global forum with real-time translation), ASCII is not sufficient. Suppose I’m developing a forum where users write in Arabic, Chinese, or Russian. If I choose ASCII, many of these characters will not display correctly, affecting both comprehension and translation. In contrast, **Unicode** allows the representation of over 140,000 characters, covering nearly all writing systems in the world. Using Unicode ensures proper rendering and data integrity, showing how choosing the right representation code can determine the success or failure of a project.

Ndjountche (2016) briefly explains other coding systems, including Gray Code and BCD. So Another important example of code is **Gray Code**, which differs from traditional binary because it only changes one bit between consecutive values. This reduces errors when reading mechanical states or movement from sensors and optical encoders, making it highly valued in such applications.

There’s also **BCD (Binary Coded Decimal)**, which encodes each decimal digit into a group of four bits. It is especially useful in **financial systems** where high precision is required, as it minimizes rounding or conversion errors, simplifies the manipulation of exact amounts, and enhances reliability in environments where numeric accuracy is critical.

600 Words

**References:**

Ndjountche, T. (2016). *Digital electronics 1: Combinational logic circuits*. John Wiley & Sons, Incorporated.

Info Pack. (2020, April 8). *Number system conversion techniques |Very easy|Fast |Decimal |Binary|Octal |Hexadecimal|*[Video]. YouTube. <https://www.youtube.com/watch?v=XXXXXXXXXX>

Bowne, S. (n.d.). *Hands-On Cryptography with Python: Leverage the power of Python to encrypt and decrypt data*. Packt Publishing.