In solving a task to translate decimal numbers into binary, octal, and hexadecimal systems, I use a representative number based on my date of birth. Now, if you take the first three numbers of my birthday (17-09-1988) you get 170. To facilitate a project focused on converting and encoding data between devices, this number will be transformed into different number systems.

Conversion Process

For example, to convert 170 to binary, I can do repeated division by 2. To do it, you divide 170 by 2 and write down the remainder (0) and then take that quotient (85) and divide it by 2 again and then write down the quotient until your quotient is 0. Stripping the remainders from the bottom up we get the binary equivalent: 10101010.

To convert to octal, I repeatedly divide 170 by 8. First, 170 ÷ 8 = 21 with remainder 2. For example, 21 ÷ 8 gives 2 remainder 5 in second step. Finally, 2: 8 = 0 with a 2 as remainder. The octal representation (read from the last remainder to the first) is therefore 252.

While converting to hexadecimal, Divide 170 by 16. The first division yields 10 with a remainder of 10, which is "A" in hexadecimal notation. Next, 10 ÷ 16 = 0, remainder 10 (or "A"). The output is AA, the hexadecimal representation of 170 (read backwards).

These conversions are necessary for encoding systems in which data must be interpreted accurately across several forms of digital media.

Importance of Number System Conversion

The ability to convert between number systems is a core skill set in tech, specifically low-level programming, data transmission, and digital electronics. Differences & Importance of

Binary as Humans we generally deal with decimal, but at the hardware level computers only can operate using binary, hence knowing how to convert from binary to human-readable formats and vice versa goes a long way in effective debugging and optimization (Mano & Ciletti, 2018). When working with memory addresses, network protocols, and machine-level instructions, developers and engineers often need to convert these values to and from binary, octal, or hexadecimal.

Hexadecimal notation is frequently used to represent color codes in web development, as well as memory addresses in debugging! If you do not have a sound understanding of these conversions, you could easily get data encoding wrong and thus cause system faults and/or miscommunication between pieces of software.

Use of ASCII, Unicode, and BCD

Not all aspects of a project will use the same coding since the data itself is different and there might be system incompatibility. ASCII works well for basic brackets in English and control characters. Unicode is ideal for internationalization and diverse, multiple languages characters and symbols. However, BCD (Binary-Coded Decimal) is useful for applications that involve decimal digits where precision is paramount, for instance, finance and accounting applications (Stallings, 2020).

For this project, ASCII could be used to log system messages, Unicode to accept user input in various languages, and BCD to display numeric values in reports. Determining the best representation requires analyzing the characteristics of the data, the target platform, and the efficiency requirements. For example, Unicode could be used in a scenario where broad support of characters is desired, while BCD would be beneficial for performing arithmetic operations without data conversion issues.

Critical Role of Correct Representation

To illustrate this with a practical example, a payment processing system that deals in multiple currencies would need to know exactly what currency each payment represents. An example is if the system only supports ASCII, it may be unable to process names and address data with accents on characters, which may lead to customer displeasure or failed transactions. The use of Unicode allows for proper handling of various languages and characters, helping maintain correct functionality and user confidence.

Comparison of Coding Schemes

Each coding representation has distinct advantages and limitations:

- **ASCII** is a 7-bit code that supports 128 characters, making it lightweight and widely compatible with older systems. However, its limited range restricts it to English and basic symbols.
- **Unicode** supports over 143,000 characters across many languages and symbols. It uses variable-length encoding (UTF-8, UTF-16), offering flexibility and international support but may consume more memory than ASCII.
- **Gray Code** is designed for minimizing errors in binary transitions, primarily used in hardware like rotary encoders rather than for textual data.

- **BCD** represents each decimal digit with a 4-bit binary code, preserving accuracy in decimal calculations. It’s ideal for financial systems but is less efficient in terms of space.
- **EBCDIC** is an 8-bit character code used mainly in IBM mainframes. It supports more characters than ASCII but suffers from limited compatibility with modern systems (Tanenbaum & Bos, 2015).

Recommendation

To perform this task, the most appropriate choice is Unicode. It has the widest compatibility, multilingual support, and is most used by modern platforms. Its versatility and flexibility make managing various types of data easy to handle in a system that must communicate with various devices, and user interfaces.

References

Mano, M. M., & Ciletti, M. D. (2018). *Digital design: With an introduction to the Verilog HDL, VHDL, and systemVerilog* (6th ed.). Pearson. https://www.pearson.com/en-us/subject-catalog/p/digital-design-with-an-introduction-to-the-verilog-hdl-vhdl-and-systemverilog/P200000003241/9780137501984

Stallings, W. (2020). *Computer organization and architecture* (11th ed.). Pearson. https://www.pearson.com/en-us/subject-catalog/p/computer-organization-and-architecture/P200000003394/9780135205129

Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson. https://www.pearson.com/en-us/subject-catalog/p/modern-operating-systems/P200000003311/9780133591620