**Name:- YASH RAJ Roll number:- 24k-0737.**

**Section 3: Arithmetic and Conversions in Number Systems**

**5. Number System Conversion and Arithmetic**

Convert 156, 1024, 255 from decimal to binary, octal, hexadecimal:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. 156 to binary  |  |  | | --- | --- | | 2 | 156 | | 2 | 78-0 | | 2 | 39-0 | | 2 | 19-1 | | 2 | 9-1 | | 2 | 4-1 | | 2 | 2-0 | |  | 1-0 |   (156)10 = (10011100)2   1. 156 to octal  |  |  | | --- | --- | | 8 | 156 | | 8 | 19-4 | |  | 2-3 |   (156)10 = (234)8   1. 156 to hexadecimal  |  |  | | --- | --- | | 16 | 156 | |  | 9-(12=C) |   (156)10 = (9C)16 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. 1024 to binary  |  |  | | --- | --- | | 2 | 1024 | | 2 | 512-0 | | 2 | 256-0 | | 2 | 128-0 | | 2 | 64-0 | | 2 | 32-0 | | 2 | 16-0 | | 2 | 8-0 | | 2 | 4-0 | | 2 | 2-0 | |  | 1-0 |   (1024)10 = (10000000000)2   1. 1024 to octal  |  |  | | --- | --- | | 8 | 1024 | | 8 | 128-0 | | 8 | 16-0 | |  | 2-0 |   (1024)10 = (2000)8   1. 1024 to hexadecimal  |  |  | | --- | --- | | 16 | 1024 | | 16 | 64-0 | |  | 4-0 |   (1024)10 = (400)16 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. 255 to binary  |  |  | | --- | --- | | 2 | 255 | | 2 | 127-1 | | 2 | 63-1 | | 2 | 31-1 | | 2 | 15-1 | | 2 | 7-1 | | 2 | 3-1 | |  | 1-1 |   (255)10 = (11111111)2   1. 255 to octal  |  |  | | --- | --- | | 8 | 255 | | 8 | 31-7 | |  | 3-7 |   (255)10 = (377)8   1. 255 to hexadecimal  |  |  | | --- | --- | | 16 | 255 | |  | 15-15 =  (FF) |   (255)10 = (FF)16 |

* Perform binary addition:

(101101 + 011011)

For doing addition first make both of same number of digits’ place

One’s place 1+1= 10 (write 0, carry 1)

Ten’s place 0+1+1 = 10 (write 0, carry 1)

Hundred’s place 1+0+1 = 10 (write 0, carry 1)

Thousandth place 1+1+1 = 10 (write 1, carry 1)

Ten thousandth place 1+0+1 = 10 (write 0, carry 1)

Final place 1+1+0 = 10

Final answer = (1001000)2

* Perform binary subtraction:

(111001 - 001001)

For doing subtraction first make both of same number of digits’ place

One’s place 1-1= 0

Ten’s place 0-0 = 0

Hundred’s place 0-0 = 0

Thousandth place 1-1 = 0

Ten thousandth place 1-0 = 1

Final place 1-0 = 1

Final answer = (110000)2

* Convert fractional binary to decimal:

To convert fractional binary number into decimal, the method is little change

First we will solve integer part then fractional part

eg. Binary number: (110.101)2

integer part

we will solve like as we do for whole binary numbers

* 110:

= (1x22) + (1x21) + (0x20)

= (1x4) + (1x2) + (0x1)

= 6 is the integer part

Now fraction part,

We will solve it by decreasing power on 2 power starts from -1 from left side and going on.

* 101:

= (1x2-1) + (0x2-2) + (1x2-3)

= (1x0.5) + (0x0.25) + (1x0.125)

= (0.5) + (0) + (0.125)

= 0.625 is fractional part

Now add these two values

= 6 + 0.625

= [ (6.625)10 = (110.101)2]

**Section 3: Arithmetic and Conversions in Number Systems**

**6. Real-World Application of Number Systems**

3-page report: about usage or using of (binary, octal, hexadecimal) in computing

In computing, different number systems such as binary, octal, and hexadecimal play crucial roles in representing and processing data. These systems serve as the foundation for machine-level operations, data storage, memory addressing, and human-computer interactions. Understanding how each system functions and where it is most applicable provides insight into the efficient workings of computers. This report compares and contrasts these number systems, discussing their advantages and limitations, along with examples of where each system is used.

Key points:

1. Binary (base-2):

(0,1), It is machine language or level 0 language,

fastest language, direct execution, no compilation.

1. Octal (base-8):

[0-7], Each digit represents 3 binary digits (bits), Easier to read and write than binary,

Used in programming, file systems, and embedded systems. Unix/Linux file permissions (e.g., 755), ASCII codes, Assembly language programming.

1. Hexa-decimal (base-16):

(0,9) and (A-F), Programming (colors, codes, memory addresses), Web development (color codes, CSS), Computer networking (MAC addresses, IP addresses), Cryptography (hashes, encryption).

|  |
| --- |
| Binary System |
| * The binary number system, also known as base-2, consists of only two digits: 0 and 1. Every number in binary is a combination of these two digits. Since computers operate using electrical signals that are either on (1) or off (0), binary is the foundation of all machine-level operations. * Binary in Machine-Level Programming:   Machine-level programming or machine code relies on binary to execute instructions directly on a computer’s hardware. Each binary digit, called a bit, represents a signal to the processor. For example, 8-bit or 32-bit architectures refer to the number of binary digits processed simultaneously.   * Example:   Consider a simple instruction like moving data from one memory location to another. In binary, the command might look something like:  10101010 11001100,  Here, each segment of the code represents an operation or memory address. Such low-level coding is tedious and prone to errors, but it is essential because this is the only form of instruction a computer’s processor can directly execute.   * Advantages of Binary:   Direct Hardware Interaction:  Since computers rely on electrical circuits, binary is the natural way to represent on/off states.   * Simplicity:   The system is simple because it only uses two digits, making it easy to represent in hardware.   * Large numbers require long binary strings, making them harder to read and understand for humans. * Prone to Errors:   Writing and debugging binary code manually can be tedious and error-prone. |

|  |
| --- |
| Octal system |
| * + The octal system, or base-8, uses digits from 0 to 7. While not as commonly used as binary or hexadecimal, octal historically played an important role in early computing. It is particularly useful for systems where grouping binary digits simplifies representation.   + **Situations Where Octal is Preferred:**   Octal is most often used in systems where binary data needs to be grouped into manageable chunks. For instance, early mainframe computers grouped binary digits into sets of three, making the octal system an easier shorthand.  Example:  Take the binary number 110101110. Grouping the digits in sets of three: 110 101 110  Now, converting each group into octal gives 6 5 6, so the binary number 110101110 is represented as 656 in octal.   * + **Advantages of Octal:**   Concise Representation: Octal reduces the length of binary numbers by grouping three bits, making the data easier to interpret.  Easier Conversion to Binary: Since octal numbers correspond directly to three bits of binary, converting between the two systems is simple.   * + **Limitations of Octal:**   Less Human-Friendly: While octal reduces binary complexity, it is still not as intuitive as decimal or hexadecimal.  Limited Use Cases: Modern computing has largely replaced octal with hexadecimal due to its more efficient grouping of bits.   * + Octal is sometimes used in computing instead of hexadecimal, perhaps most often in modern times in conjunction with file permissions under Unix systems. It has the advantage of not requiring any extra symbols as digits (the hexadecimal system is base-16 and therefore needs six additional symbols beyond 0–9). |

|  |
| --- |
| Hexadecimal |
| * + In modern computing, memory addresses are often very large numbers, making binary representations cumbersome. Hexadecimal shortens these addresses by grouping four bits into one hex digit. This makes memory addresses more readable and manageable.   Example:  Consider the 32-bit binary number 10101111011000011010100110111011. In hexadecimal, this can be represented as: AF61A9BB  This significantly reduces the complexity when dealing with large binary values, particularly in tasks like debugging or analyzing memory addresses.   * + **Advantages of Hexadecimal:**   Compact Representation: Hexadecimal compresses large binary values into shorter, readable forms.  Easy Conversion: Like octal, hexadecimal is directly convertible to binary by grouping bits (in this case, four bits per hex digit).  Widely Used: It is the preferred system for memory addressing, color codes in web design (e.g., #FFFFFF for white), and debugging machine code.   * + **Limitations of Hexadecimal:**   Less Intuitive for Humans: Even though it is more readable than binary, hexadecimal is not as easily understood by the average user compared to decimal.  Requires Conversion: Unlike decimal, which is naturally understood by humans, hexadecimal requires knowledge of conversion between binary or decimal systems.   * + **Real-World Applications:**   1. Computer graphics and game development  2. Network protocol analysis  3. Cryptocurrency and blockchain  4. Embedded systems and microcontrollers |