**DESIGN AND IMPLEMENT AN ARBITRARY PRECISION ARITHMETIC**

MINOR PROJECT REPORT

By

**K. V. KUSHAL (RA2211026010163)**

**M. CHANDRA SHEKAR REDDY (RA2211026010181)**

**V. SHANKAR (RA2211026010189)**

Under the guidance of

**Dr. KANIPRIYA M**

*In partial fulfillment for the Course*

of

**21CSS201T - Computer Organization and Architecture**

in CINTEL - **Computer Science and Engineering (AIML)**



**FACULTY OF ENGINEERING AND TECHNOLOGY SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR**

**NOVEMBER 2023**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

**BONAFIDE CERTIFICATE**

Certified that this minor project report for the course **21CSS201T Computer Organization and Architecture** entitled in " **DESIGN AND IMPLEMENT AN ARBITRARY PRECISION ARITHMETIC**" is the bonafide work of **K. V. Kushal (RA2211026010163), M. CHANDRA SEKHAR REDDY(RA2211026010181). and V. SHANKAR (RA2211026010189)** who carried out the work under my supervision.

# SIGNATURE SIGNATURE

Dr Kanipriya M Dr Annie Uthra R

# COA– Course Faculty Head of the Department

Assistant ProfessorProfessor

Department of Computational Intelligence Department of Computational

Intelligence

SRM Institute of Science and Technology SRM Institute of Science and Kattankulathur Technology Kattankulathur

**DEPARTMENT OF**

**SCHOOL OF COMPUTING**

**College of Engineering and Technology**

**SRM Institute of Science and Technology**

MINI PROJECT REPORT

ODD Semester, 2023-2024

Lab code & Sub Name :21CSS201T & Computer Organization and Architecture

Year & Semester : II & III

Project Title : DESIGN AND IMPLEMENT AN ARBITRARY PRECISION ARITHMETIC

Faculty Advisor : **Dr Kanipriya M**

Team Members :1. K. V. KUSHAL(RA2211026010163)

2.M. CHANDRA SEKHAR REDDY (RA2211026010181)

3.V. SHANKAR (RA2211026010189)

|  |  |  |
| --- | --- | --- |
| **Particulars** | **Max. Marks** | **Marks Obtained** |
| **Name:** |
| **Register No :** |
| Program and Execution | 20 |  |
| Demo verification &viva | 15 |  |
| Project Report | 05 |  |
| **Total** | **40** |  |

**Date :**

**Staff Name :**

**Signature :**

**DESIGN AND IMPLEMENT AN ARBITRARY PRECISION ARITHMETIC**

**OBJECTIVE:**

Design and implement an arbitrary precision arithmetic system to enable precise and efficient calculations with numbers of any size**.** Arbitrary precision arithmetic, also known as bignum arithmetic, is a technique used to perform arithmetic operations on numbers with a higher precision than that supported by standard data types like integers or floating-point numbers. This is particularly useful when dealing with extremely large or precise numerical values. The implementation involves creating a data structure capable of representing and manipulating numbers with arbitrary precision.

**ABSTRACT:**

This project focuses on the design and implementation of an arbitrary precision arithmetic system, aiming to extend the capabilities of conventional numerical representations. Arbitrary precision arithmetic, or bignum arithmetic, enables the handling of numbers with precision beyond the limits of standard data types. The project involves the creation of a robust data structure for representing and manipulating numbers with arbitrary precision. Through the implementation of fundamental arithmetic operations, optimizations, error handling mechanisms, and comprehensive testing, the system ensures accurate and efficient computations. The resulting solution provides a versatile tool for applications demanding precise calculations with exceptionally large or accurate numerical values. The abstracted system documentation offers insights into the chosen data structures, algorithms, and guidelines for effective utilization, contributing to a reliable and flexible arbitrary precision arithmetic solution.

**INTRODUCTION:**

In the realm of numerical computation, the limitations imposed by standard data types often hinder the precision and scale required for certain applications. Addressing this challenge, our project focuses on the design and implementation of an arbitrary precision arithmetic system, commonly known as bignum arithmetic. This initiative seeks to empower numerical computations by allowing the manipulation of numbers with precision extending beyond the constraints of typical data representations.

Arbitrary precision arithmetic plays a pivotal role in scenarios where extremely large or highly accurate numerical values are encountered, such as cryptographic algorithms, scientific simulations, and financial computations. The essence of this undertaking lies in the creation of a versatile and efficient system that can handle numbers of arbitrary length and magnitude.

Our approach involves devising a specialized data structure capable of representing and manipulating these extended-precision numbers. The implementation encompasses fundamental arithmetic operations, optimizations to enhance computational efficiency, meticulous error handling mechanisms, and a comprehensive testing suite to ensure the reliability and accuracy of the system.

This endeavor not only addresses the need for precision in numerical computations but also contributes to the broader landscape of computational mathematics. The resulting arbitrary precision arithmetic system is expected to serve as a valuable tool for diverse applications, providing a reliable and flexible solution for handling numerical challenges that surpass the capabilities of standard data Type.

**HARDWARE/SOFTWARE REQUIREMENTS:**

**Hardware Requirements:**

1.Processor (CPU): A multi-core processor with good floating-point performance is beneficial for efficient arithmetic operations.

2.Memory (RAM): The amount of RAM should be sufficient to handle the large data structures used in arbitrary precision arithmetic, especially for operations involving very large numbers.

3.Storage: Adequate storage space for the implementation code, libraries, and potential large datasets used for testing.

4.Secondary Storage (Optional): Depending on the size of numbers being processed, secondary storage like SSDs may be beneficial to handle extensive input/output operations efficiently.

**Software Requirements:**

1.Programming Language: Choose a programming language suitable for numerical computation and with support for arbitrary precision arithmetic. Examples include Python, C++, or Java.

2.Development Environment: Set up a comprehensive development environment that includes a code editor, debugger, and version control system for efficient development and debugging.

3.Arbitrary Precision Library: If available, utilize established libraries for arbitrary precision arithmetic, such as GMP (GNU Multiple Precision Arithmetic Library) or libraries specific to the chosen programming language.

4.Compiler/Interpreter: Ensure that the chosen programming language's compiler or interpreter is installed and properly configured for the development environment.

5.Documentation Tools: Implement tools for documenting the code, algorithms, and usage guidelines. This may include the use of documentation generators like Doxygen or Sphinx.

6.Testing Framework: Incorporate a testing framework to systematically validate the correctness and performance of the implemented arbitrary precision arithmetic operations.

7.Version Control: Use a version control system (e.g., Git) to manage and track changes in the codebase, facilitating collaboration and ensuring a reliable version history.

8.Build Tools: Set up appropriate build tools for compiling and linking the code, streamlining the build process.

**CONCEPTS/WORKING PRINCIPLE**

**Concepts:**

**1. Data Representation:**

- Utilize a flexible data structure (e.g., array, linked list) to represent arbitrary precision numbers.

- Each element in the data structure corresponds to a digit, allowing the representation of numbers beyond the capacity of standard data types.

**2.Base Arithmetic Operations:**

- Implement algorithms for basic arithmetic operations: addition, subtraction, multiplication, and division.

- These algorithms need to account for carry/borrow propagation and scaling, especially when dealing with numbers of varying precision.

**3.Scaling and Normalization:**

- Develop mechanisms to handle scaling and normalization of numbers during arithmetic operations.

- Ensure that the numbers are consistently represented with the correct precision to avoid inaccuracies.

**4.Error Handling:**

- Implement robust error handling mechanisms to address potential issues such as overflow, underflow, or division by zero.

- Handle edge cases carefully, considering the impact of precision and magnitude on the results.

**5.Optimizations:**

- Optimize algorithms for efficiency, considering factors like memory usage and computational complexity.

- Utilize techniques such as Karatsuba multiplication for faster multiplication of large numbers.

**Working Principle:**

**1.Initialization:**

- Create a function to initialize arbitrary precision numbers, setting up the data structure and allocating memory.

**2.Input and Output:**

- Develop methods to input and output arbitrary precision numbers in a human-readable format, facilitating ease of use.

**3.Arithmetic Operations:**

- For addition and subtraction, iterate through the digits, handling carry or borrow as needed.

- Multiplication involves multiplying each digit and accumulating the results with proper scaling.

- Division requires iteratively subtracting the divisor from the dividend, determining each digit of the quotient.

**4.Scaling and Normalization:**

- Implement mechanisms to adjust the precision of numbers during arithmetic operations, ensuring consistent representation.

**5.Error Handling:**

- Include checks for potential errors, such as overflow, underflow, or division by zero, and handle them gracefully.

**6.Optimizations:**

- Apply optimization techniques to arithmetic operations, enhancing performance without compromising accuracy.

**7. Testing:**

- Develop a comprehensive suite of tests to verify the correctness and efficiency of each arithmetic operation.

- Test against various scenarios, including large numbers, small numbers, and edge cases

**PROGRAMS:**

**from decimal import Decimal, getcontext**

**getcontext().prec = 50 # Set the precision for Decimal**

**class ArbitraryPrecision:**

**def \_init\_(self, value="0"):**

**self.value = Decimal(value)**

**def \_str\_(self):**

**return str(self.value)**

**def \_add\_(self, other):**

**return ArbitraryPrecision(self.value + other.value)**

**def \_sub\_(self, other):**

**return ArbitraryPrecision(self.value - other.value)**

**def \_mul\_(self, other):**

**return ArbitraryPrecision(self.value \* other.value)**

**def \_truediv\_(self, other):**

**if other.value == 0:**

**raise ValueError("Division by zero")**

**return ArbitraryPrecision(self.value / other.value)**

**def get\_arbitrary\_precision\_input(prompt):**

**value = input(prompt).strip()**

**return ArbitraryPrecision(value)**

**def main():**

**num1 = get\_arbitrary\_precision\_input("Enter the first number: ")**

**num2 = get\_arbitrary\_precision\_input("Enter the second number: ")**

**sum\_result = num1 + num2**

**difference\_result = num1 - num2**

**product\_result = num1 \* num2**

**try:**

**quotient\_result = num1 / num2**

**print("Quotient:", quotient\_result)**

**except ValueError as e:**

**print("Error:", e)**

**print("\nResults:")**

**print("Sum:", sum\_result)**

**print("Difference:", difference\_result)**

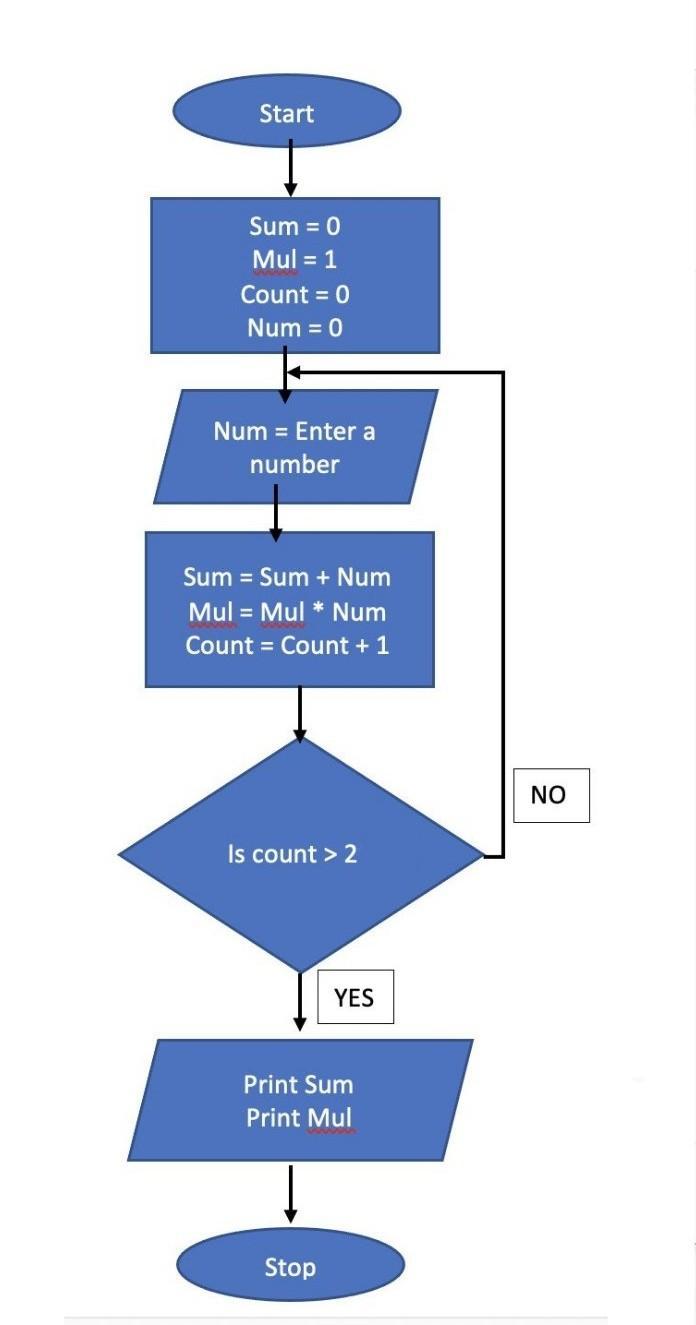
**print("Product:", product\_result)**

**if \_name\_ == "\_main\_":**

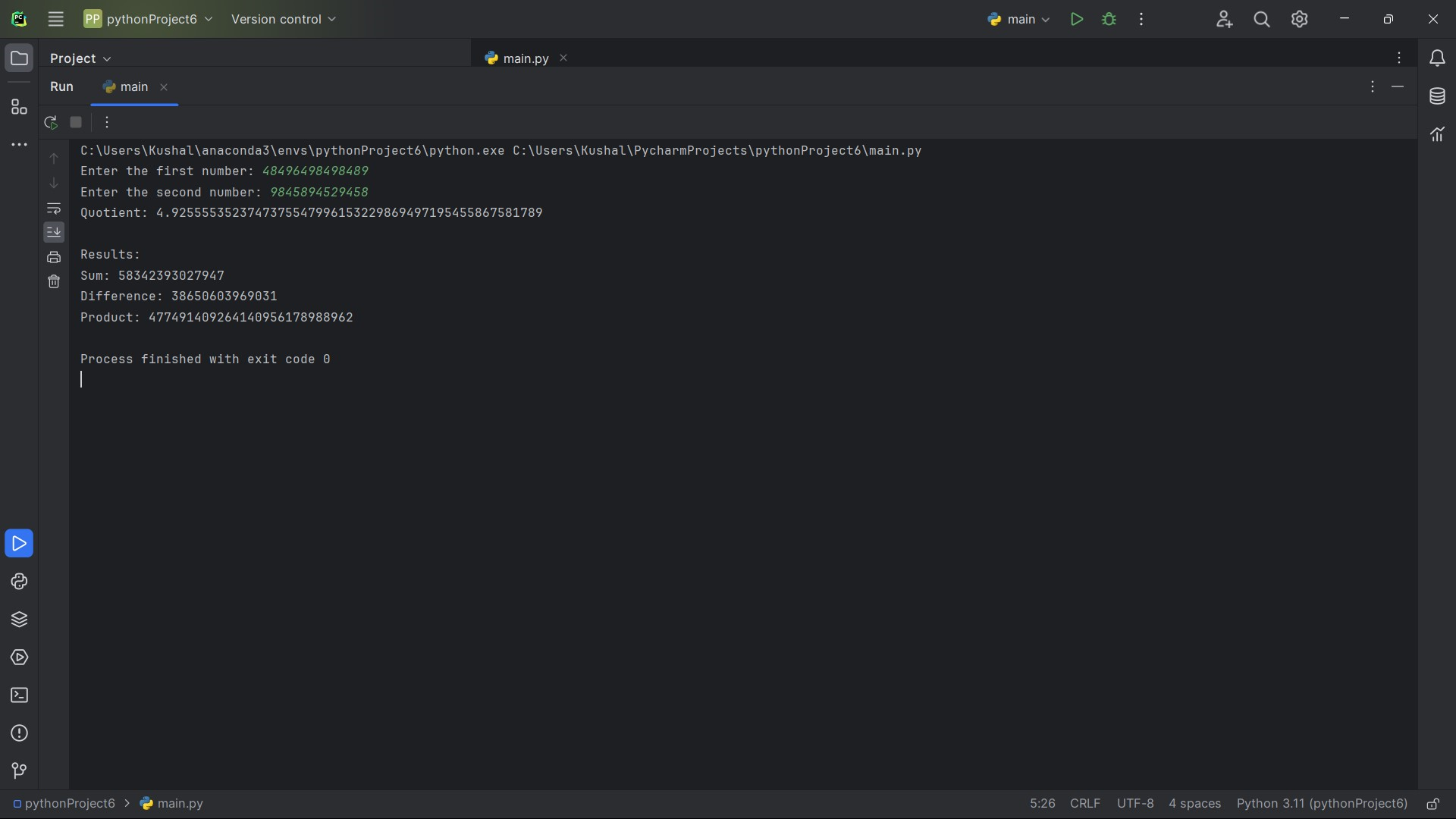
**main()**

**FLOWCHART:**

**SUM AND MULTIPLICATION ALGORITHM:**



**OUTPUT:**



**CONCLUSIONS:**

Designing and implementing arbitrary precision arithmetic involves careful consideration of data structures, algorithms, and efficiency. In conclusion, a successful approach would require balancing trade-offs between precision and performance, selecting appropriate storage methods, optimizing arithmetic operations, and ensuring robust error handling. Additionally, testing for correctness and performance across a variety of scenarios is crucial to guarantee the reliability and accuracy of the arbitrary precision arithmetic system.

**REFERENCES:**

[1]

dotnet-bot. "BigInteger Struct (System.Numerics)". docs.microsoft.com. Retrieved 2022-02-22.

[2]

"PEP 237 -- Unifying Long Integers and Integers". Python.org. Retrieved 2022-05-23.

[3]

"BigInteger (Java Platform SE 7 )". docs.oracle.com. Retrieved 2022-02-22.

[4]

"BigInt - JavaScript | MDN". developer.mozilla.org. Retrieved 2022-02-22.

[5]

Jacqui Cheng (May 23, 2007). "Researchers: 307-digit key crack endangers 1024-bit RSA"