DESIGN AND IMPLEMENTATION OF SLOW AND FAST DIVISION ALGORITHM

ABSTRACT

Division is a fundamental arithmetic operation performed in computer architecture. In this abstract, we present an overview of two division algorithms: the slow division algorithm and the fast division algorithm, implemented in a computer architecture context.

I.INTRODUCTION

Division is a fundamental arithmetic operation that involves dividing one number (the dividend) by another (the divisor) to obtain the quotient and remainder. In computer architecture, efficient division algorithms are crucial for performing division operations quickly and accurately. Two commonly used division algorithms are the slow division algorithm (long division) and the fast division algorithm (restoring division).

The division algorithm is a traditional approach that is widely taught and understood. It follows a step-by-step process of repeatedly subtracting the divisor from the dividend until the remainder becomes less than the divisor. The quotient is determined by counting the number of subtractions performed, and the final remainder is the remaining value after the subtraction process.

II.LITERATURE SURVEY

"High-Speed Division Algorithms: A Survey" by G.D. Akkarakaran and M. Antonakakis (1996)

This paper provides a comprehensive survey of high-speed division algorithms, including both slow and fast division algorithms. It covers various techniques and optimizations employed in the design of division algorithms, along with their performance analysis and comparison.

"Efficient Algorithms for Dividing Polynomials and Power SerieAlthough primarily focused on polynomial division, this classic work by Donald Knuth discusses the concepts and design principles behind division algorithms, including slow and fast division algorithms. It provides a theoretical background and analysis of the algorithms' efficiency.

"Arithmetic Division Algorithms and Implementations in Modern Processors" by R. Ranjan and N. Das (2012)

This paper presents a detailed survey of division algorithms implemented in modern processors. It covers various division techniques, including both slow and fast division algorithms, and discusses their implementation details, performance characteristics, and trade-offs.

"Fast Division and Square Root for Processors" by J.-M. Muller and A. Tisserand (2001)

This paper focuses on fast division algorithms and their implementation in processors. It presents different fast division algorithms, including restoring division, and discusses their advantages and drawbacks.

III.OBJECTIVES

Slow Restoring Division Algorithm:

1.Understanding: Gain a thorough understanding of the principles and steps involved in the slow restoring division algorithm (long division).

2.Accuracy: Perform division operations accurately by correctly determining the quotient and remainder.

3.Conceptual Clarity: Develop a clear understanding of the digit-by-digit processing and iterative subtraction approach employed in slow division.

4.Implementation: Implement the slow restoring division algorithm in hardware or software to perform division operations efficiently.

5.Education: Use slow division as a teaching tool to help students grasp the fundamental concepts of division and strengthen their mathematical skills.

Fast Restoring Division Algorithm:

1.Efficiency: Improve the efficiency of division operations by reducing the number of iterations and minimizing the number of subtraction operations.

2.Speed: Perform division operations quickly, especially for large numbers, by utilizing bitwise shifting and binary comparison techniques.

3.Hardware Implementation: Design and implement efficient hardware architectures for fast restoring division in computer processors and arithmetic units.

4.Trade-offs and Optimizations: Explore different architectural choices and optimizations to enhance the performance of fast division algorithms.

5.High-Performance Computing: Enable faster division operations in high-performance computing systems where computational efficiency .

IV.OUTCOMES

1.Efficiency Improvements: By carefully designing and optimizing the hardware or software implementation of the slow and fast restoring division algorithms, designers can achieve significant efficiency improvements in division operations.

2.Resource Utilization: Designing efficient division algorithms involves considerations of resource utilization, such as minimizing hardware components, memory usage, or computational complexity. By optimizing the algorithm design, designers can achieve better resource utilization and potentially reduce the hardware cost or power consumption of the system.

3.Speed Enhancement: The design of fast restoring division algorithms aims to achieve faster division operations compared to slow division. Successful design outcomes will result in accelerated division speeds, which can have a significant impact on the overall performance of applications that heavily rely on division computations.

4.Accuracy and Precision: An essential outcome of algorithm design is maintaining accuracy and precision in division operations. Designers need to ensure that the division algorithms produce correct results with minimal or no rounding errors or loss of precision, especially when dealing with floating-point or high-precision arithmetic.

5.Hardware Implementation: Designing division algorithms for hardware implementation involves considerations of architecture, circuit design, and component integration. A successful outcome will be the development of efficient and compact hardware units or dedicated processing modules capable of executing slow or fast restoring division with high throughput and low latency.

6.Trade-off Analysis and Optimization: Designing division algorithms requires making trade-offs between factors such as performance, accuracy, resource utilization, and implementation complexity. Outcome measures can include well-informed design choices, effective trade-off analysis, and optimization strategies to strike a balance between conflicting design objects.

V.CHALLENGES

1.Execution Time: Slow restoring division algorithms can be time-consuming, especially for large numbers. The iterative nature of the algorithm, involving multiple subtractions, can result in longer execution times, impacting overall system performance.

2.Digit-Level Processing: Slow restoring division requires processing digits sequentially, which can be challenging to optimize for efficient hardware implementation. Dealing with carry propagation, digit selection, and managing intermediate results pose design challenges.

3.Memory Utilization: Slow restoring division may require additional memory for storing intermediate results, quotient digits, and temporary variables. Efficient memory management becomes crucial, especially in resource-constrained systems or when dealing with large operands.

4.Numerical Precision and Rounding Errors: Slow division algorithms can introduce rounding errors or loss of precision due to repeated subtractions. Ensuring accurate results while managing numerical precision and minimizing rounding errors can be challenging.

5.Algorithm Complexity: Fast restoring division algorithms involve bitwise shifting, binary comparisons, and subtraction operations, making the algorithm design complex. Handling various scenarios, such as negative numbers, overflow conditions, and edge cases, can be challenging.

6.Circuit Design: Implementing fast restoring division in hardware circuits requires careful consideration of circuit design aspects. Efficient circuitry for shifting, subtraction, and comparison operations, along with proper handling of timing constraints and propagation delays, can be challenging.

7.Quotient Digit Selection: Fast restoring division algorithms require efficient methods for selecting quotient digits during each iteration. Determining the optimal quotient digit selection technique is a challenge, considering factors like accuracy, performance, and resource constraints.

8.Optimization Trade-offs: Achieving optimal performance in fast restoring division algorithms requires exploring various optimization techniques. Balancing trade-offs between hardware complexity, performance gains, and power consumption is a challenge.

VI.ARCHITECTURE

1.Divisor and Dividend Registers: These registers store the divisor and dividend values during the division process. They hold the input values and are used for digit-by-digit processing.

2.Quotient and Remainder Registers: These registers store the partial quotient and remainder values during the division process. They are updated in each iteration and accumulate the results.

3.Subtraction Unit: This component performs the subtraction operation between the current dividend and the divisor, updating the remainder and determining the quotient digit for each iteration.

4.Comparator: The comparator unit compares the current remainder with the divisor to determine if the subtraction result is positive or negative. This information is used to set the quotient digit and decide subsequent actions.

5.Control Unit: The control unit coordinates the sequencing of operations in the division process. It generates control signals to enable/disable components, control register updates, and handle carry propagation.

6.Iteration Counter: This component keeps track of the number of iterations performed in the division process. It helps control the termination condition and keeps the track of the division progress.

7.Multiplexers and Logic Gates: Multiplexers are used to select the appropriate inputs for various operations, such as shifting, subtraction, and digit selection. Logic gates perform the necessary logical operations to control the flow of data and control signals.

The architecture for implementing fast restoring division algorithm in computer systems involves several key components:

1.Divisor and Dividend Registers: Similar to the slow division architecture, these registers hold the divisor and dividend values during the division process. They store the input values and are used for bitwise shifting and comparisons.

2.Quotient Register: This register holds the quotient value during the division process. It accumulates the quotient digits as they are determined in each iteration.

3.Shifter Unit: The shifter unit performs bitwise left shifting of the dividend and the quotient in each iteration. It handles the shifting of bits to create the next set of dividend bits for comparison.

4.Subtraction Unit: The subtraction unit subtracts the divisor from the shifted dividend to determine if the result is positive or negative. This information is used to set the quotient digit.

5.Comparator: The comparator compares the result of the subtraction with zero to determine the sign and determine the next actions in the division process.

6.Control Unit: The control unit coordinates the sequencing of operations in the fast division algorithm. It generates control signals to enable/disable components.

7.Iteration Counter: The iteration counter keeps track of the number of iterations performed in the division process. It helps control the termination condition and tracks the progress of the division.

VII.HARDWARE AND SOFTWARE MODEL FOR IMPLEMENTATION

1.Arithmetic Logic Unit (ALU): The ALU performs arithmetic and logical operations, including subtraction, shifting, and comparison operations required for both slow and fast restoring division algorithms.

2.Registers: Dividend, divisor, quotient, and remainder registers store the values used in the division process. These registers hold the operands and intermediate results during the computation.

3.Control Unit: The control unit manages the sequencing of operations, controls the flow of data between components, and generates control signals to coordinate the operation of the hardware units.

4.Multiplexers and Logic Gates: Multiplexers are used to select inputs for various operations, such as digit selection, shifting, and subtraction. Logic gates perform the necessary logical operations to control data flow and generate control signals.

5.Clock Generator: The clock generator provides the necessary clock signal to synchronize the operations of the hardware components

VII.CONCLUSIONS

In conclusion, the slow and fast division algorithms plays a significant roles in computer architecture , providing different approaches to perform division operation accurately.

The slow division also knows as long division, follows a step by step process. While it is conceptually straight forward, it can be time consuming , especially for larger numbers.

On the other hand, the fast division,it optimizes the division process by utilizing bitwise shifting and subtraction operations.