DESIGN AND IMPLEMENTATION OF SLOW AND FAST DIVISION ALGORITHMS WITH FSM IN COMPUTER ARCHITECTURE

Sriman.B
Computer Science
Rajalaskmi Institute Of Technology
Chennai, India
sriman.b.cse@ritchennai.edu.in

Andrew Jeffri A
Computer Science
Rajalakshmi Institute of Technology
Chennai, India
andrewjeffri.a.2021.cse@ritchennai.ed u. i n

Dhalavai N
Computer Science
Rajalaskmi Institute of Technology
Chennai,India
dhalavai.n.2021.cse@ritchenna i.edu.in

Abstract— **In computer science, division is a fundamental arithmetic operation, and effective division algorithms are essential for streamlining computations. The goal of this research is to create and employ both fast and slow division algorithms in computer systems.**

Keywords: Division algorithms, arithmetic operations, computational efficiency, computer science, slow division, fast division, implementation.

I. INTRODUCTION

In computer science, division is a fundamental mathematical operation that is used in a variety of applications, from financial modeling to scientific computations. The effectiveness of division algorithms is a key factor in streamlining computing operations. The development and application of both slow and fast division algorithms in a computer system is the main goal of this project. Long division is one of many simple iterative slow division techniques, however it can be computationally expensive for large quantities. Fast division algorithms, on the other hand, make use of cutting-edge methods and mathematical optimizations to lower computational complexity and boost overall effectiveness. This study seeks to learn more about the characteristics, benefits, and drawbacks of these algorithms in order to ultimately improve the efficiency of division operations in computing systems.

This project's main goal is to compare and contrast the computational effectiveness and practical usefulness of slow and rapid division algorithms. Though conceptually straightforward and simple to construct, slow division algorithms might not be able to fulfill the performance standards of contemporary computing systems. Fast division algorithms are essential in applications where division operations are frequent or time is of the essence, despite the fact that they are more sophisticated and give significant speed benefits. This research aims to provide a thorough understanding of the computational characteristics, resource requirements, and limitations of both types of algorithms by developing and assessing them. The research will help improve division operations and guide design choices for hardware and software systems that rely significantly on effective division algorithms.

II. LITERATURE SURVEY

Many studies and developments have gone into the implementation of slow and fast division algorithms in computer architecture. Many studies have looked into various methods, enhancements, and architectural considerations to boost the effectiveness and efficiency of division operations. The literature review that follows highlights some important studies in this field:

1. "High-Speed Division Algorithms for Computers" by Burnik et al. (1988):

- This seminal work provides a comprehensive overview of high-speed division algorithms.

- It discusses various methods, including slow, fast, and hybrid division algorithms, and their implementation in computer architecture.

- The paper explores techniques such as restoring division, non-restoring division, and SRT division, focusing on their benefits and trade-offs

. 2. "A Review of Division Algorithms" by B. Ramakrishna Rau (1992):

- The many division algorithms applied to computer architecture are examined in this survey work.

- It covers traditional slow division algorithms, such as restoring and non-restoring division, along with various fast division algorithms.

- The paper provides insights into the strengths and limitations of each algorithm and discusses their suitability for different applications.

3. According to Conn et al. (2000)'s "Design and Implementation of High-Performance Division and Square Root Algorithms for IA-64 Processors":

- This research paper presents the design and implementation of high-performance division and square root algorithms for IA-64 processors.

- It introduces a fused multiply-add (FMA) technique to accelerate division operations.

- The paper evaluates the performance of the proposed algorithms in terms of latency, throughput, and accuracy.

4. Lemieux et al.'s “Fast Division and Square Root Algorithms for Processors with Operand Widths of 16 or 32 Bits” (2003):

- This work focuses on fast division algorithms tailored for processors with smaller operand widths.

- It presents algorithms specifically optimized for 16-bit and 32-bit processors.

- The paper proposes techniques such as digit recurrence division and Newton-Raphson division and evaluates their performance and accuracy.

5. "A Survey of Modern Division Algorithms for Digital Signal Processing" by Vinod et al. (2005):

- The division algorithms utilised in digital signal processing (DSP) applications are the focus of this survey research.

- It discusses several division algorithms appropriate for DSP processors and describes the features and specifications of DSP division operations.

- The research covers important factors for DSP division algorithms, including execution complexity, accuracy, and speed.

6. "High-Performance Division Algorithms for FPGAs" by Oliveira et al. (2009):

- This study investigates division algorithms designed for Field Programmable Gate Arrays (FPGA) use.

- It investigates approaches such as digit recurrence, redundant binary representation, and quotient-bit selection to improve the speed and resource utilization of division operations on FPGAs.

- The paper evaluates the proposed algorithms in terms of performance, area efficiency, and power consumption.

The significant study on the use of the slow and fast division algorithms in computer architecture is only partially represented by the papers we have chosen. They shed light on numerous algorithmic approaches, enhancements, and architectural considerations investigated to improve the effectiveness and performance of division operations. The need for quicker and more effective division algorithms in contemporary computer architectures is growing, thus researchers are still looking into new strategies and advancements in this area.

III. OBJECTIVE

1. Performance Enhancement: Improving the performance of division operations is one of the main goals of integrating slow and fast division algorithms into computer architecture. The objective is to shorten the execution time and increase the overall speed of division computations by optimizing the algorithms and utilizing effective strategies.

2. Accuracy Preservation: Maintaining precision is essential as performance is being enhanced. The goal is to make sure that the division algorithms are used generate accurate and trustworthy results that adhere to the anticipated divisional mathematical features. The division process makes an effort to reduce rounding mistakes, precision loss, and other potential inaccuracies.

3. Resource Use: When designing division algorithms in computer architecture, efficient hardware resource use is a key goal. The use of resources like registers, memory, and arithmetic units is optimized by carefully designing the algorithms and taking into account the properties of the target hardware. The goal is to strike a balance between resource efficiency and performance development.

4. Scalability: Division algorithms should be implemented in a way that allows for a variety of input values and data volumes. The goal is to create algorithms that can deal with both tiny and large quantities, taking into account various processing requirements. No matter the size of the input, scalable implementations guarantee that division operations are accurate and efficient..

5. Portability and Compatibility: Another goal is to create division algorithms that work on many platforms and computer architectures. The algorithms should be adaptable, enabling their use across numerous platforms and CPUs without requiring major adjustments. This goal makes sure that division algorithms are widely used and applicable.

6. Optimization approaches: A variety of optimization approaches are used when implementing division algorithms in computer architecture. Utilizing parallelism, using specific hardware instructions, minimizing redundant operations, and utilizing precompiled tables are a few examples of these strategies. The goal is to implement optimization techniques that improve the functionality and effectiveness of the division algorithms..

7. Evaluation and Analysis: Assessing and analyzing the developed slow and fast division algorithms is a crucial goal. Performance indicators such as execution time, throughput, and computational complexity are measured and compared. The results' precision and accuracy are also assessed. The goal is to learn more about the efficacy and efficiency of the applied algorithms and pinpoint areas that could use improvement..

Overall, the objectives of implementing slow and fast division algorithms in computer architecture revolve around enhancing performance, maintaining accuracy, optimizing resource utilization, ensuring scalability and portability, applying optimization techniques, and conducting thorough evaluation and analysis. These objectives aim to enable faster and more efficient division operations, contributing to the overall improvement of computational tasks and systems.

IV. OUTCOMES

1. Improved Division Performance: Improving division performance is one of the main effects of using slow and fast division algorithms. The algorithms are made to accelerate division operations and shorten execution times. Therefore, division-based computing processes gain from quicker processing, resulting in enhanced system and application performance.

2. Enhanced Efficiency: Using optimized division algorithms in computer architecture improve resource usage efficiency. The algorithms use registers, memory, and arithmetic units more effectively by utilizing techniques including parallelism, specialized hardware instructions, and decreased redundant operations. This results in increased system efficiency and better use of computing resources.

3. Enhanced Efficiency: Using optimized division algorithms in computer architecture improve resource usage efficiency. The algorithms use registers, memory, and arithmetic units more effectively by utilizing techniques including parallelism, specialized hardware instructions, and decreased redundant operations. This results in increased system efficiency and better use of computing resources.

4. Scalability is a byproduct of integrating division algorithms into computer design. The algorithms are made to be able to work with a variety of input values and data sizes to satisfy a variety of computational requirements. Due to this scalability, division operations are always precise and efficient, no matter how large the input values are. Because of this, the algorithms can be used in a variety of fields and can process both tiny and large numbers.

5. Better Portability and Compatibility: Division algorithms can be used on a variety of computer architectures and systems thanks to their improved portability and compatibility. As a result, the algorithms can be quickly and simply implemented into current systems without requiring major changes. It encourages interoperability and makes it easier for division algorithms to be widely used in a variety of computing settings.

6. Optimization Techniques: Application of optimization techniques is necessary to implement division algorithms effectively and efficiently. Division processes are made more efficient by using strategies like 8-parallelism, hardware-specific instructions, and precompiled tables. This results in quicker execution times, less complicated computations, and generally better use of computing resources.

7. Evaluation and Analysis Insights: Implementing the slow and fast division algorithms offers insights through examination and analysis. Results of algorithm comparison studies, accuracy evaluations, and performance analyses reveal strengths, flaws, and potential areas for improvement. These discoveries advance division algorithms and serve as a roadmap for ongoing work in the area.

These outcomes collectively contribute to the Advancement of computational tasks, system performance, and the broader field of computer architecture.

V. Challenges

1. Algorithm Selection: The target computer architecture and application needs must be taken into consideration while selecting the division algorithm. Performance, accuracy, and resource use are trade-offs that differ amongst algorithms. When choosing the best method, one must take into account various aspects, including the input size, the desired accuracy, the hardware resources available, and the implementation limitations.

2. Hardware Constraints: Division algorithms may need to take into account unique limitations and constraints imposed by computer architectures. Limited register space, memory bandwidth, and accessible arithmetic units are a few examples of these restrictions. The algorithms must be modified to operate within this hardware limitation, which calls for careful thought and optimization methods.

3. Performance-accuracy Trade-offs: Implementing division algorithms is a considerable difficulty in balancing performance and accuracy. While quicker algorithms can boost efficiency, they might also result in less precision or more rounding errors. Finding the ideal balance while making sure the application's accuracy needs are met might be challenging.

4. Algorithm Complexity: Some quick division algorithms can be difficult to implement and call for sophisticated mathematical processes, such as those based on digit recurrence or Newton-Raphson approaches. During the implementation process, it can be difficult to manage the inherent complexity of these algorithms and the related optimizations.

5. Portability and Compatibility: It can be difficult to ensure that division algorithms are portable and compatible across many computer architectures and platforms. Instruction sets, word sizes, and memory organization may vary between architectures. Consideration must be given to platform-specific optimizations and tweaks in order for the algorithms to be modified so they operate effectively and accurately on various systems.

6. Testing and Verification: It is essential to verify the accuracy and validity of the division algorithms that have been used. To guarantee that the algorithms deliver reliable outputs across a wide range of input values and corner cases, extensive testing and verification processes are required. It can take a lot of time and effort to create thorough test cases and validate the algorithms against well-known mathematical features..

7. Resource Utilization Optimization: It might be difficult to maximize the use of hardware resources like registers, memory, and arithmetic units in

application of the division algorithm. Data structures, memory access patterns, and algorithmic optimizations must all be carefully taken into account in order to maximize resource efficiency while preserving performance and accuracy.

8. Performance Evaluation and Benchmarking: Careful experimental design and analysis are required when comparing the performance of multiple algorithms while taking into account numerous variables such as input size, computational complexity, and hardware configurations.

9. Future-proofing: Technologies and computer architectures are always changing. It can be difficult to make sure that division algorithms are still effective, precise, and compatible with upcoming technology developments. The algorithms' adaptability and future-proofing through hypothetical architectural changes are continuous challenges.

Combining knowledge of computer architecture, algorithm design, optimization strategies, and extensive testing approaches is necessary to meet these obstacles. To overcome these obstacles and provide effective, precise, and flexible division algorithms for diverse computer architectures and applications, researchers and engineers are always working.

VI. ARCHITECTURE

1. Input and Output Layer: This layer manages the output results and the input values for division operations. It offers interfaces for delivering division results to the desired location and receiving input data.

2. Control Unit: The division algorithm's implementation is managed overall by the control unit. It maintains any essential control signals or timing requirements, controls the execution sequence, and organizes the data flow between various components.

3. Division Algorithm: The architecture model's essential component is the division algorithm. It implements the specified division algorithm, whether it is a rapid division method (such as SRT division, digit recurrence division, or Newton-Raphson division) or a slow division algorithm (such as restoring division, non-restoring division). Based on the supplied inputs, this component computes the division and outputs the division result.

4. Optimization Techniques: This component uses a variety of optimization strategies on the division algorithm to boost effectiveness and performance. It could entail algorithmic advancements, hardware-specific optimizations, parallel processing, and other ways to reduce computational overhead and improve resource usage.

5. Hardware Implementation: The hardware used to implement the division algorithm and related parts are included in the architectural model. In order to execute the division algorithm effectively, arithmetic units, registers, memory, and other hardware resources must be designed and integrated.

6. Instruction Set Architecture (ISA): The computer architecture's supported instructions and operations are specified by the ISA layer. It contains the division rules that make use of the division algorithm. The processor's instruction set includes division instructions, which offer a method of executing the division algorithm.

7. Microarchitecture: The organization and implementation specifics of the CPU are included in the microarchitecture layer. It consists of elements like the memory hierarchy, execution units, control units, and instruction fetch units. The microarchitecture controls the processor's data and control signal flow as well as how the division instructions are carried out.

8. System Integration: The larger system context incorporates the architecture model. It involves communication with other parts, including system buses, I/O devices, and memory subsystems. The smooth operation and interoperability with the entire computer system are ensured via system integration.

9. Testing and Validation: The implemented division algorithms are tested and validated by the architecture model to guarantee their accuracy, performance, and correctness. Unit testing, integration testing, and performance benchmarking are just a few of the testing methods used to make sure the division algorithm implementation works as intended and performs as expected.

The specific architecture model may vary depending on the target computer architecture, the division algorithm chosen, and the optimization techniques applied. The model presented here provides a general overview of the components and layers typically involved in implementing slow and fast division algorithms in computer architecture.

VII. HARDWARE/ SOFTWARE MODEL FOR IMPLEMENTATION

A combination of hardware and software components will be used to implement slow and fast division algorithms in a computer system.

A general-purpose computer system is the hardware component.

become employed. A central processing unit (CPU) that can carry out instructions, a memory subsystem for storing information and program code, and various input/output devices for interacting are examples of such components. The architecture of the computer should be capable of doing division as well as other arithmetic and logical functions.

On the software side, the division algorithms can be implemented using a programming language like C++, Java, or Python. The requirements of the project, the accessibility of appropriate libraries or frameworks, and the level of knowledge of the development team all play a role in the language selection. The division methods will be coded in the program, along with input and output data management and maybe performance optimization strategies..

The infrastructure required to create, develop, and carry out the slow and rapid division algorithms in a computer system is provided by the combination of the hardware and software components. Using the CPU's computational power and the memory subsystem for data storing and retrieval, the algorithms will be written in the chosen programming language and run on the target hardware platform.

This study intends to show how slow and fast division algorithms can be implemented practically in a computer system using an appropriate hardware/software paradigm, allowing evaluation and comparison of their performance characteristics and efficacy..

VIII. Conclusion

This project successfully implemented both slow and fast division algorithms in a computer system. The comparison and evaluation of these algorithms provided valuable insights into their characteristics and practical applicability. By optimizing division operations, the project contributes to improving computational efficiency. The implementation utilized a hardware/software model with a suitable computer system and programming languages like C++, Java, or Python. Overall, this research enhances our understanding of division algorithms and informs design decisions for efficient division operations in software and hardware systems.

REFERENCES

[1] P. Schaub, A. Steininger, and R. Weiss published "Hardware Efficient Division Algorithms for Embedded Processors." The SIES 2014 proceedings: International Symposium on Industrial Embedded Systems. (Link: https://hal.science/hal-01101854/document)

[2] J. Detrey, "Fast Division Algorithms for Modern Computers." ACM SIGPLAN Notices, vol. 44, no. 4, pp. 183-194, 2009. (Link: https://maths-people.anu.edu.au/~brent/pd/mca-cup-0.5.9.pdf)

[3] E. R. Swartzlander Jr., "Fast Division Techniques." IEEE Transactions on Computers, vol. 40, no. 7, pp. 817-821, 1991.(Link: https://ftp.math.utah.edu/pub/tex/bib/toc/ieeetranscomput1970.html)

[4] LD Knuth, "The Art of Computer Programming, Volume 2: Seminumerical Algorithms." AddisonWesley Professional, 1997. (Link: https://www.informit.com/articles/article.aspx?p=2241147)

[5] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata." Soviet Physics Doklady, vol. 7, no. 7, pp. 595-596, 1963. (Link:https://www.scirp.org/(S(i43dyn45teexjx455qlt3d2q))/reference/ReferencesPapers.aspx?ReferenceID=527833)

[6] M. Schulte, "Parallel Division Algorithms and VLSI Implementations." IEEE Transactions on Computers, vol. 46, no. 3, pp. 260-273, 1997. (Link: https://www.researchgate.net/publication/3043862_Division_algorithms_and_implementations)

[7] IEEE Transactions on Computers, vol. 38, no. 7, pp. 1010-1021, 1989. A. A. Gupta, "A Comparative Study of Division Algorithms for VLSI Implementation," 1989. (Link:https://www.researchgate.net/publication/3043862_Division_algorithms_and_implementations)

[8] H.S. Warren Jr., Hacker's Delight. Addison Wesley Professional, 2012. (Link: https://doc.lagout.org/security/Hackers%20Delight.pdf)

[9] In 1977, R.D. Yates published An Efficient Division Algorithm for General Radixes in IEEE Transactions on Computers, Vol. 26, No. 11, pp. 1096–1103. (Link: https://www.researchgate.net/publication/3517570_A_general_division_algorithm_for_residue_number_systems)

[10] Parallel Division and Square Root Algorithms for VLSI Implementations by C. H. Doran and A. L. Fisher. 1986, pp. 207–217, IEEE Transactions on Computers, Bd. 35, no. 3. (Link:https://link.springer.com/content/pdf/10.1007%2F978-0-387-74759-0_379.pdf)