Division Algorithms Elucidate: Scrutinising Slow and Fast Division Algorithms

Sriman.B
Professor ,Computer science and Engineering
Rajalakshmi Institute Of Technology
Chennai, India
Sriman.b@ritchennai.edu.in

Anuradha S
Computer science and Engineering
Rajalakshmi Institute Of Technology
Chennai, India
anuradha.s.2021.cse@ritchennai.edu.in

Aathineha R
Computer science and Engineering
Rajalakshmi Institute Of Technology
Chennai, India
aathineha.r.2021.cse@ritchennai.edu.in

Archana G S
Computer science and Engineering
Rajalakshmi Institute Of Technology
Chennai, India
archana.g.s.2021.cse@ritchennai.edu.in

***Abstract* -- The paper discusses numerous optimisation methods in a division algorithm that uses the numerator (N) and the Denominator (D) as inputs, and Euclidean division is used to calculate their quotient and remainder. These algorithms vary in a variety of ways, including the quotient convergence rate, computational formulations, and fundamental hardware primitives; this enables us to select the ideal algorithm. In slow division algorithms such as restoring and SRT division, the divisor is repeatedly subtracted from the partial remainder until the desired quotient is obtained. Fast division, on the other hand, uses non-restoring algorithms like the Newton-Raphson and Goldschmidt algorithms to cut down on the number of rounds needed. The design and implementation of both slow and fast division algorithms utilising the Mealy and Moore model are the main topics of this work, and the performance, functionality, latency, throughput, resource utilisation, speed, and complexity of the division algorithm can be analysed. This concludes the study on slow and fast division algorithms in terms of their performance, including throughput, execution time, and hardware complexity, which clarifies the accuracy and speed of the two separate algorithms and also advances computer mathematics and benefits today's processor.**

Keywords – Efficiency, Computational time, Division Algorithm , restoring, Computer Architecture.

I.INTRODUCTION

In the domain of computer architecture and the design of modern calculators, the elemental arithmetic operation used is division. It is very challenging to implement it efficiently. There are two different approaches to solving the division operation, namely the slow division algorithm and the fast division algorithm. According to the given problem, the best solution can be used; each has its own pros and cons. By Exploring the Elementary of the Division algorithm the Researches gained insights into more complex and efficient Division algorithms used in evolving modern computing systems. This Optimization improves the Performance and ease the implementation.

A. Slow Division Algorithm

The basic approach to performing straightforward division operations is the slow division algorithm, which can be implemented in computer software and hardware. It is a tedious and procedural algorithm, as outlined. The first step is initialization, in which the algorithm initialises the divisor and the dividend as input. The next step is comparison, in which the dividend and divisor are compared; if the divisor is greater than the dividend, then the result, which is the quotient, is zero. If the divisor is less than dividend, it moves to the next step, which is the division loop, where the algorithm subtracts the divisor from the dividend in each iteration. The upcoming step is the quotient calculator, which calculates the quotient after the termination of the division loop.

The Final step is the remainder calculation, where the remainder is calculated by subtracting the product of the quotient and the divisor from the dividend. As an output, the algorithm returns the values of the quotient and remainder. As the name suggests, its performance is very slow as it undergoes multiple subtractions in a loop that consumes more time. Therefore, to improve its efficiency, computation time, and performance, the fast division algorithm was introduced in modern computing. It is used in educational and resource environments as it is simplified and resource-efficient.

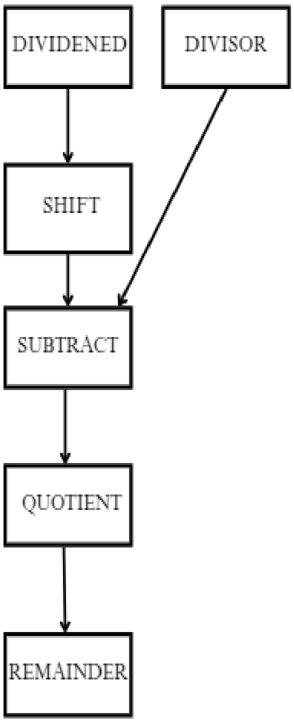

Fig.1: Block Diagram for Slow Division Algorithm.

B. Fast Division Algorithm

As the traditional method is time-consuming and not that efficient, to overcome this challenge, the fast division algorithm gained insight. They reduce the number of iterations, which reduces time consumption and increases efficiency in applications, which benefits both the hardware and software. As the evolution of technologies continues, the algorithm gets more efficient. The number of iterations is reduced by the process called convergence, where the iterations are performed until the desired level of precision is reached; this is also called the termination condition. This introduces some level of approximation compared to the traditional method, which is precise. Increased computational speed, enhanced system responsiveness,improved throughput, energy efficiency, scalability, compatibility with modern architectures, and wide applicability are all results of using a fast division algorithm in computer architecture. These results aid in the effective and high-performance execution of division operations, making it possible for computer systems to be quicker and more responsive across a variety of applications and domains. Utilising a fast division algorithm in computer design aims to maximise division operations' speed and efficiency while retaining reasonable levels of accuracy.

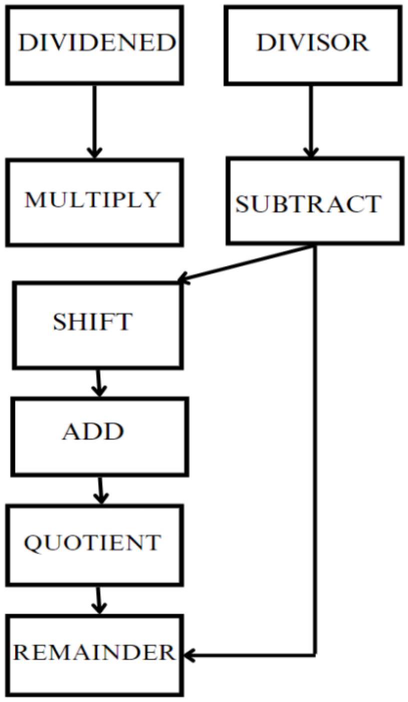

Fig.2: Block Diagram for Fast Division Algorithm

II.TYPES OF FAST DIVISION ALGORITHM

In comparison to conventional division algorithms, fast division algorithms are intended to carry out division operations more quickly and often with less computing complexity. Several such rapid division algorithms are listed below:

1.radix-2 and radix-4 division algorithm

The radix-2 and radix-4 division algorithms make use of the binary representation of numbers to carry out division in parallel. These algorithms use parallel processing to handle several bits of the dividend and divisor at once, which allows for quicker division operations. Radix-4 division methods improve much further by employing base-4 representations and working in quadrants.

2.SRT

Although it can be used for division operations, the SRT division algorithm, as was already indicated, is mostly employed for process scheduling. The responsiveness is increased while the overall execution time is decreased since it chooses the partial quotient with the shortest remaining execution time in each iteration.

3. Newton-raphson

Division can be performed using the Newton-Raphson algorithm, which was first developed for locating equation roots. It employs multiplications to perform division and approximates the reciprocal of the divisor, which can be faster than traditional division algorithms. It is very helpful when dividing by a constant because it is possible to precalculate the reciprocal.

4. Non restoring algorithm

An effective method for integer division is the non-restoring division algorithm that was previously stated. It reduces computing effort by avoiding restoring the divisor throughout each loop. It is frequently used in hardware implementations because of how quick and easy it is.

5. Barrett's algorithm

Barrett's division algorithm uses precomputed constants to speed up division operations and is based on modular arithmetic. It works especially well when multiplying by a constant divisor. Compared to traditional approaches, Barrett's division requires fewer iterations and can produce a faster division.

6. Montgomery multiplication

A method for modular multiplication called Montgomery multiplication can also be used for division. The division action is transformed into a series of multiplications by Montgomery-based division algorithms, which can be executed more quickly. Due to their speed and resistance to specific attacks, these algorithms are frequently used in cryptography applications.

These are a few illustrations of fast division algorithms that perform better or have less computational complexity than conventional division techniques. The selection of an algorithm is influenced by a number of variables, including the type of divisor, the necessary level of efficiency, and the particular situation in which operations for divisions are carried out.

DIVISON ALGORITHM	ADVANTAGE	LIMITATION
SLOW DIVISION ALGORITHM	1.Simplified Implementation	1.More Computation Time
	2.Resource-efficient	2.Less Efficient
	3.More precise	3. Dependency on Operand Sizes
	4.Ease of Understand	4. Interrupt Handling
	5.Minimal hardware or software required	5. Complex Control Logic
FAST DIVISION ALGORITHM	1.Efficient	1. Trade-off between Speed and Accuracy
	2. High speed and accuracy	2. Complexity and Development Effort
	3. High Optimisation	3. Adaptability to Varying Inputs
	4.Less computation Time	4. Iterative Approximation Error
	5. Flexibility	5. Hardware Constraints

Fig.3:.Comparative Analysis Of Slow and Fast Division Algorithm.

III. TYPES OF SLOW DIVISION ALGORITHM

Compared to speedier division algorithms, slow division algorithms have a higher level of computational complexity and may call for more repetitions. They are typical or traditional methods of performing division. Some examples of slow division algorithms are as follows:

1.Long Division

The technique of long division is one that is frequently taught and comprehended. It entails carrying down digits from the payout and conducting repeated subtraction in order to calculate the quotient. When compared to other division algorithms, long division is comparatively slower since it takes numerous steps and sequential processing.

2. Restoring division

Iteratively restoring the divisor is a key component of the method known as "restoring division." To find the quotient and remainder, it performs a series of subtraction and shifting operations. In comparison to algorithms that don't use restoring division, restoring division requires more computational steps to recover the divisor, which may slow down performance.

3. Non restoring division (with correct step)

As previously indicated, restoring division algorithms can often be slower than non-restoring division algorithms. Negative remainders must be handled by an additional correction step in some non-restoring division algorithms, though. Because of the additional computations introduced by this corrective phase, the execution may be slower.

4. Division by repeated subtraction

Divide a sum by the dividend as many times as necessary until the dividend equals the divisor; this is the simplest method of division. One digit of the quotient is represented by each subtraction operation. In comparison to other division techniques, this algorithm is slower since it requires more repetitions and has a larger computational complexity.

Slow division algorithms are frequently used when speed is not the main issue or when working with numbers that cannot be handled effectively by faster division methods. In situations where performance is not crucial, they could nonetheless be helpful in teaching the fundamentals of division or in other applications. Faster division algorithms are preferred to achieve efficient computation, but in many real-world situations.

IV. MEALY AND MOORE MODEL

A. Moore Model

The Moore model, also known as the Moore machine or Moore armature, is a theoretical model used in computer armature and automata propositions. It's named after the American computer scientist Edward F. Moore. In the environment of computer architecture, the Moore model refers to a type of finite state machine (FSM) that consists of a set of countries, a set of inputs, a set of labours, and a transition function.

The crucial specific of the Moore model is that the labours depend only on the current state of the machine rather than on both the current state and the input. One important specificity of the Moore model is that it's memoryless. The coming state of the machine depends only on the current state and the input, but it doesn't have access to any history or former inputs. This makes the Moore model suitable for operations where the outcome is solely determined by the current state.

B. Mealy Model

A Mealy model is a type of Finite State Machine (FSM) that describes the behaviour of a system. In the context of computer architecture, Mealy models can be used to represent controllers or data paths in a processor. A Mealy model is defined by a set of states, inputs, outputs, and transition rules. Each state represents a particular configuration or state of the system, and inputs are signals or events that can trigger state transitions.

Outputs represent the actions or results that the system produces in each state. In computer architecture, inputs to Mealy models can include signals such as opcodes, control signals, and clock pulses. Outputs can include control signals for various components of the processor, such as ALUs (arithmetic logic units), registers, memories, and I/O devices.

We here use mealy model to elucidate the slow and fast division algorithm. Below is the model and

Transition table for both the algorithm implemented with mealy model. The Transition table explains the present state and transition to the next state with the input variable. By understanding the Automata we can gain insight in both the algorithms with clear distinct of there implementation and effects on efficiency in performing the division operation.

1 .FSM For Fast Division Algorithm

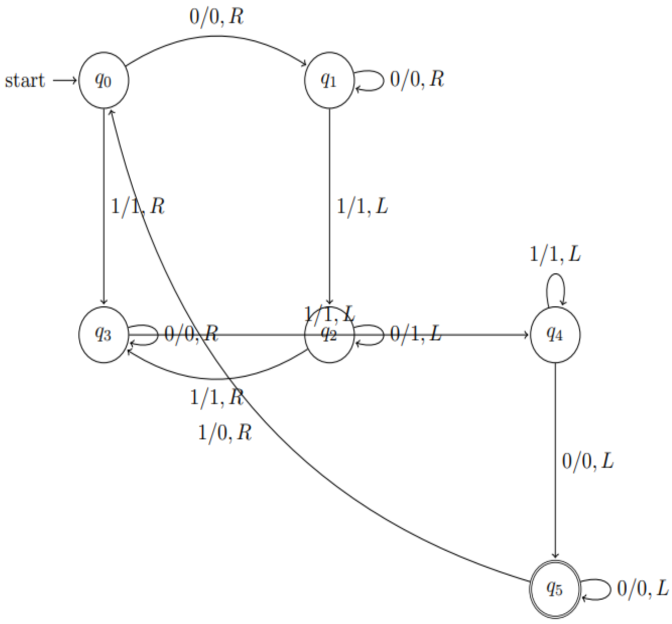

Fig.4: Finite State Machine for Fast Division Algorithm.

State	Input	Output	Action
q_0	0	0	R
q_0	1	1	R
q_1	0	0	R
q_1	1	1	L
q_2	0	1	L
q_2	1	1	R
q_3	0	0	R
q_3	1	1	L
q_4	0	0	L
q_4	1	1	L
q_5	0	0	L
q_5	1	0	R

Fig.5:Transition table for fast division FSM

2. FSM for Slow Division Algorithm

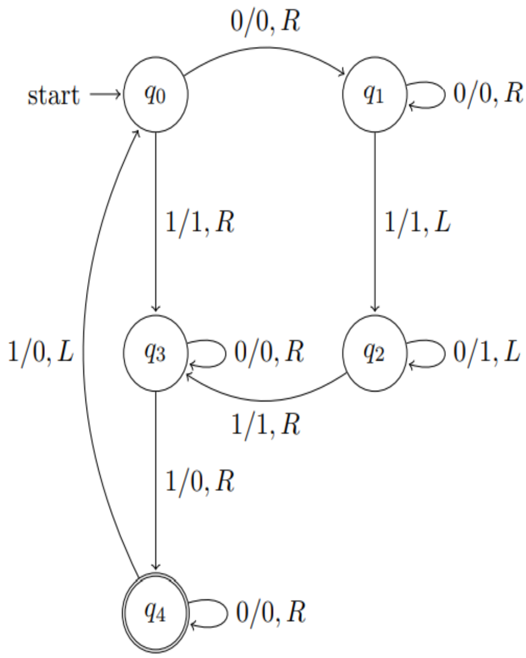

Fig.6: Finite State Machine For Slow Division Algorithm.

State	Input	Output	Action
q_0	0	0	R
q_0	1	1	R
q_1	0	0	R
q_1	1	1	L
q_2	0	1	L
q_2	1	1	R
q_3	0	0	R
q_3	1	0	R
q_4	0	0	R
q_4	1	0	L

Fig.7.Transition Table for the Slow Division FSM

V. IMPLEMENTATION OF ALGORITHMS IN VERILOG

A. Implementation of Slow Division Algorithm

```
Module slow_ division
 (
  Input [31:0] dividend,
  Input [31:0] divisor,
  Output reg [31:0] quotient,
  Output reg [31:0] remainder,
  Output reg done
);
 Reg [63:0] accumulator;
  Reg [31:0] count;
  Always @(posedge clk) begin
   If (!done) begin
   If (accumulator[31:0] >= divisor) begin
       Accumulator[63:0] = accumulator[63:0] – divisor;
       Quotient[count] = 1;
   End
      Else
       Quotient[count] = 0;
       Accumulator = {accumulator[62:0], quotient[count], dividend};
  Count = count + 1;
   If (count == 32)
       Done = 1;
   End
  End

Initial begin
   Accumulator = {32'b0, dividend};
   Count = 0;
   Done = 0;
  End
 End module
```

B. Implementation of Fast Division Algorithm

```
Module fast_ division
(
  Input [31:0] dividend,
  Input [31:0] divisor,
  Output reg [31:0] quotient,
  Output reg [31:0] remainder,
  Output reg done
);
  Reg [31:0] count;
  Reg [31:0] subtractor;
  Reg [31:0] temp_ remainder;

Always @(posedge clk) begin
    If (!done) begin
     If (count == 32) begin
       Done = 1;
       Return;
     End

 If (subtractor >= 0) begin        Temp_ remainder = temp_ remainder << 1;
       Temp_ remainder[0] = dividend[31];
       Dividend = dividend << 1;
     End
     Else begin
       Temp_ remainder = temp_ remainder << 1;
       Temp_ remainder[0] = ~dividend[31];
       Dividend = (dividend << 1) + 1;
     End

 Subtractor = dividend – divisor;
 If (subtractor >= 0) begin
       Dividend = subtractor;
       Quotient[count] = 1;
     End
```

```
Else begin
  Quotient[count] = 0;
End
  Count = count + 1;
End
End

Initial begin
  Count = 0;
  Subtractor = dividend;
  Temp_ remainder = 0;
  Done = 0;
End
End module
```

VI.ACKNOWLEDGEMENT

I would like to express my Special Thanks of Gratitude to my college mentor “Mr.Sriman.B” who gave us the golden opportunity to this wonderful project on this INTEL , which helped me in doing lot of research and I came to know about so many new things I am really thankful to them. Secondly I would also like to thanks my team members for finalizing this project within the limited time frame.

VII.CONCLUSION

Both slow and fast approaches must be considered when designing and implementing division algorithms for computer architectures. The focus of slow division algorithms is to ensure accuracy and precision in division calculations, which often require repeated subtraction or shift operations. This algorithm guarantees reliable results, but can be computationally expensive, especially for large numbers. Fast division algorithms, on the other hand, prioritize speed and efficiency by employing techniques such as digit repetition and non-restoring division. These methods provide faster results, but may sacrifice accuracy or require additional hardware resources. Therefore, the choice between a slow division algorithm and a fast division algorithm depends on the specific needs of the application, considering factors such as speed, accuracy, and available hardware resources. Balancing these factors ensures optimal design and

VIII.REFERENCE

[1] Bhoyar R., Palsodkar P. and Kakde S. 2015. Design and implementation of Goldschmidts algorithm for floating point division and square root. In: IEEE International Conference on Communications and Signal Processing. pp. 1588-1592.

[2] S. F. Oberman and M. J. Fiynn. 1997. Division algorithms and implementations. IEEE Transaction on Computers. 46(8): 833-854.

[3] N. Sorokin. 2006. Implementation of high-speed fixed-point dividers on FPGA. Journal of Computer Science and Technology. 6(1): 8-11.

[4] Ercegovac M. D. and Muller J. M. 2005. Variable radix real and complex digit-recurrence division. In: IEEE International Conference on ApplicationSpecific Systems, Architecture and Processors. pp. 316-321.

[5] Floating-Point Working Group. 1987. IEEE standard for binary floating-point arithmetic. In: ACM Special Interest Group on Programming Languages. pp. 9-25.

[6] S. Kaur, Suman, M. S. Manna and R. Agarwal. 2013. VHDL implementation of non-restoring division algorithm using high speed adder/subtractor. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering. 2(7): 3317-3324.

[7] Harris D. L., Oberman S. F. and M Horowitz. A. 1997. SRT division architectures and implementations. In: 13th IEEE Symposium on Computer Arithmetic. pp. 18-25.

[8] Jain S., Pancholi M., Garg H. and Saini S. 2014. Binary division algorithm and high speed deconvolution algorithm. In: 11th IEEE International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology

[9] F. Obennan and M. 1. Flynn, "Design issues in division and other floating-point operations," IEEE Transactions on Computers, vol. 46, pp. 154-161, 1997.

[10] S. F. Obennan and M. J. Flynn, "Division algorithms and implementations," IEEE Transactions on Computers, vol. 46, pp. 833854, 1997.

[11]Nicolas Boullis and Arnaud Tisserand, “On digit-recurrence division algorithms for self-timed circuits”, in Research Report published at Institut National De Recherche En Informatique Et En Automatique, France, 2012.

[12] Peter Soderquist and Miriam Leeser, "Division and square root: choosing the right implementation," IEEE Micro, vol. 17, no. 4, pp. 56-66,

[13] Implementation of Non Restoring Division Algorithm Using High Speed Adder/Subtractor.