International Journal of Advanced Technology in Engineering and Science
Vol. No. 12, Issue No. 10, October 2024
www.ijates.com
ijates
ISSN 2348 - 7550

Enhanced Computational Algorithm for Binary Division Using the Comparison Method

Mamta Shree[1], Sunil Suman[2]

[1]*Research Scholar Department of Mathematics Magadh University Bodh Gaya, Bihar*

[2]*Professor Deptt. of Mathematics, Magadh University Bodh Gaya, Bihar*

ABSTRACT

An arithmetic circuit performs binary division as its fundamental operation. It is less complex than decimal division because the outcome is consistently either 1 or 0. Binary division expresses all values of the dividend, divisor, quotient, and remainder as solely 1 and 0. A variety of binary division methods exist, including the restoring method, the non-restoring method, division using the XOR logic operation, and the SRT division and comparison method. This study introduces a novel notion of the comparative division approach. The comparison division technique facilitates rapid computing and enhances system performance.

Keywords: *Binary division, restoring method, comparison technique, non-restoring method, and magnitude comparator*

1. INTRODUCTION

Binary division is a method that demonstrates the frequency with which the divisor D divides the dividend A. When the dividend exceeds the divisor, we subtract the divisor from the dividend and assign a value of 1 to the quotient bit in division. The computer system executes the subtraction operation using 2 complement representation. Division is the repeated subtraction of the divisor from the dividend until the remainder is less than the divisor. The division algorithm operates on the following principles [1]:

A. If the dividend segment that exceeds the divisor is greater than or equal to the divisor,
- Subsequently, deduct the divisor from the dividend and
- Place the result of the subtraction "1" at the right end of the quotient.
- If the outcome is zero, append “0” to the right end of the quotient.

B. Shift the divisor one position to the right.

C. Continue iterating until the dividend is smaller than the divisor and the quotient is accurate. The dividend is the residual amount.

This technique, if executed straightforwardly, is exceedingly time-consuming. Aligning the leading digits of the divisor and dividend before the initial subtraction significantly expedites the process, and shifting the divisor one position to the right occurs whenever the partial residual is less than the divisor before the shift. A shift may be required prior to any subtraction.

There exist two types of division algorithms: digit recurrence division and division by convergence. Digit recurrence division is straightforward and exhibits lower complexity than the convergence division technique. Three methods exist for division algorithms.

a) The restoration procedure involves the use of supplementary restoration cycles
b) Comparison Method
c) Non-restoring approach (restoration cycles eliminated)

The non-restoring division algorithm is the most efficient of the digit recurrence division methods as it eliminates the necessity for restoration cycles.

2. METHOD FOR RESTORATION

If you simply use the division algorithm, it can be time-consuming as it requires you to check the remainder with the divisor after each subtraction. When you divide by three digits, the restoring division formula is the easiest to use. In restoring division, subtraction keeps going until the sign of the partial remainder changes. As soon as the sign shifts, we immediately add the divisor and decrease the increasing quotient. This happens before the right shift. If the temporary partial remainder is less than zero, the restoring division performs two additions per iteration. This makes the worst case wait longer.

2.1 Things about the Restoring Method:

a. The process of subtraction keeps going until the partial remainder changes signs.
b. Directly add the divider.
c. Additional rounds of restoration are needed for the restoration.
d. The value's sign changes when an action occurs in the restoring method.
e. Adds the divider right away.

3. A METHOD THAT DOESN'T RESTORE

Using this non-restoring method [2] is extremely fast, as each cycle only involves one addition. For this, you need the fraction set {+1,-1}. We use the number +1 for removal and the number -1 for addition [3].

3.1 Things about the non-restoring method

a) When the sign changes, there is a shift, followed by one or more increases, until the sign changes again.
b) When the sign changes, there is a shift, followed by one or more increases, until the sign changes again.
c) If the radix is negative, the quotient would require a conversion routine to return to its usual form. [2]
d) Each step only performs one addition or subtraction. If the sign of the partial remainder and the division are the same, we set the sign bit in the quotient to 0.
e) The length of the operand (divisor) influences the sequence counter's number.

4. METHOD OF COMPARISON

There are two ways to apply the comparison method to a division algorithm:

a) Divide with small tasks
b) Division with a comparator for size

International Journal of Advanced Technology in Engineering and Science
Vol. No. 12, Issue No. 10, October 2024
www.ijates.com
ijates
ISSN 2348 - 7550

4.1 Division with small operations

The comparison method compares the divisor and reward before the subtraction step. We subtract the divisor from the dividend if it is greater than or equal to the dividend. Nothing happens when the dividend is less than the divisor. Instead, we move the partial leftover to the left and perform another comparison of the numbers. You can find out what the comparison operation is before the subtraction operation by looking at how the addition operation ends in a parallel adder. Let us say that register A holds the dividend and register B holds the divisor. It is possible to do the comparison with EA - A+B+1. If E is less than 0, the dividend is less than the divisor. If E is greater than 1, the dividend is more than the divisor. The comparison process compares A and B before subtracting them. If A is greater than or equal to B, we subtract b from A. There is no action for A<B. There are three parts to the hardware implementation: output carries, partial remainder bits, and quotient bits.

↓ *Quotient Bit is inserted here*

Output carry	Partial remainder	Quotient bits

Fig (1) Program division

To retain the carry bit after the addition operation, we use the end carry flip-flop. Examining the terminal carry initiates the comparison process. To calculate the results, first validate the register length to ensure there is no overflow condition. We assume that the dividend must be less than the divisor. The comparison approach accelerates division operations by eliminating the time required for restoring partial remainders. [1] [4].

4.1.1 Algorithm for the comparative method:

Initially, verify the divide overflow condition by summing the two's complement of register B with register A. Next, assign the resultant value to registers E and A, and initialize the quotient bit to 0. Only a left shift operation occurs if the end carry flip-flop E has a value of 0. We execute the subtraction and left shift operations if E holds a value of 1. We assign a value of 1 to the quotient bit.

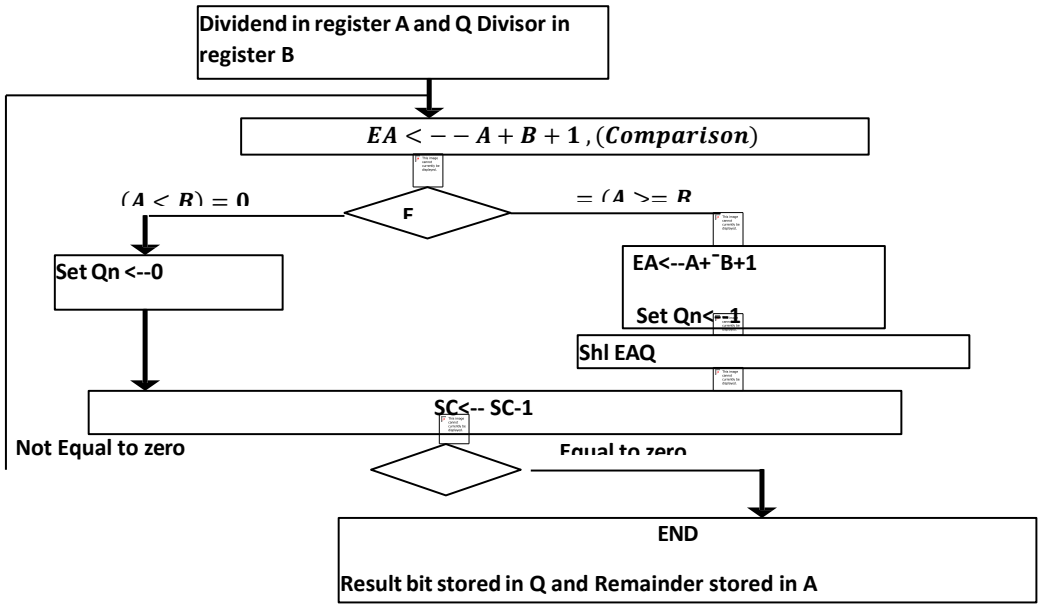

Fig (2) Flow chart for comparison algorithm

This comparison approach involves only two operations: addition and left shift. You can enhance software and hardware components by substituting certain divisions with shifts, additions, or subtractions to optimize compilers, and you can replace hardware dividers with simpler adders in VLSI circuits.

4.1.2 Attributes of the comparison algorithm:

Minimize execution duration when using a non-negative radix, division operations don't require a supplementary set. Each step involves only two micro-operations: addition and left shift. Reduced circuitry is necessary. This approach enhances performance due to its rapid speed. However, when the dividend is smaller than the divisor, the lack of restoration might result in an insufficient residual.

5. PROPOSED WORK: COMPARISON ALGORITHM UTILIZING MAGNITUDE COMPARATOR

Evaluating the equality of two binary values is a frequently employed process in computer systems and device interfaces. We can minimize the comparator's hardware by using only two outputs, from which we can derive the third output. For example, an NOR gate can derive the EQ output if both the LT and GT outputs are available. Consequently, when both the GT and LT outputs are zero, the third output (i.e., EQ) is one. Another method for the division algorithm involves initially comparing the magnitudes of both integers using a magnitude comparator. This method conserves processing time when A equals B, setting the quotient bit to 1, hence eliminating the necessity for any micro-operation. When A is greater than B, we first normalize the dividend to be less than the divisor to meet the necessary condition. We can execute the addition and subtraction micro-operation when A exceeds B and when A falls below B. The magnitude comparator circuit conserves initial processing time for comparison operations, hence enhancing speed. In situations where A is greater than or equal to B, the magnitude comparator circuit performs the necessary addition, subtraction, and shift micro-operations, which are similar to those in restoring and non-restoring methods [5]. This concept's advantage is in its avoidance of doing micro-operations for $A = B$ to produce the quotient bit, eliminating the necessity to verify the sequence counter value. Both the restoring and non-restoring methods rely on micro-operations to verify the initial equivalence condition, which requires the execution of numerous micro-operations to generate the quotient bit, even when $A = B$.

The magnitude comparator comparison algorithm possesses the following characteristics:

a) The initial comparison conserves computational time.
b) The pace is rapid compared to other division methods.
c) Incorporating the combinational circuit raises the cost.

5.2 Steps of the Algorithm

1. Verify the requisite condition for the division operation, namely that the dividend must be less than the divisor for both fixed and floating-point data representations. The normalizing process ensures that the dividend is less than the divisor in floating-point data encoding. This method executes operations incrementally. However, the quotient bit receives a value of 1 if A equals B.

The dividend reduces the divisor if A is greater than B.

If A is less than B, we push the dividend to the left and set the quotient bit to 0.

If A equals B, we subtract the divisor; if the remainder is 0, we set the quotient bit to 1, perform no bitwise operations, and conclude the procedure.

Integrating the magnitude comparator with micro-operations enhances processing speed and reduces execution time by using fewer computer instructions. This technique provides high-performance computing methods crucial for addressing the growing demand for computation. The aim of the Division of Computational Algorithms is to attain the performance and reliability standards necessary for essential computational scientific applications [6].

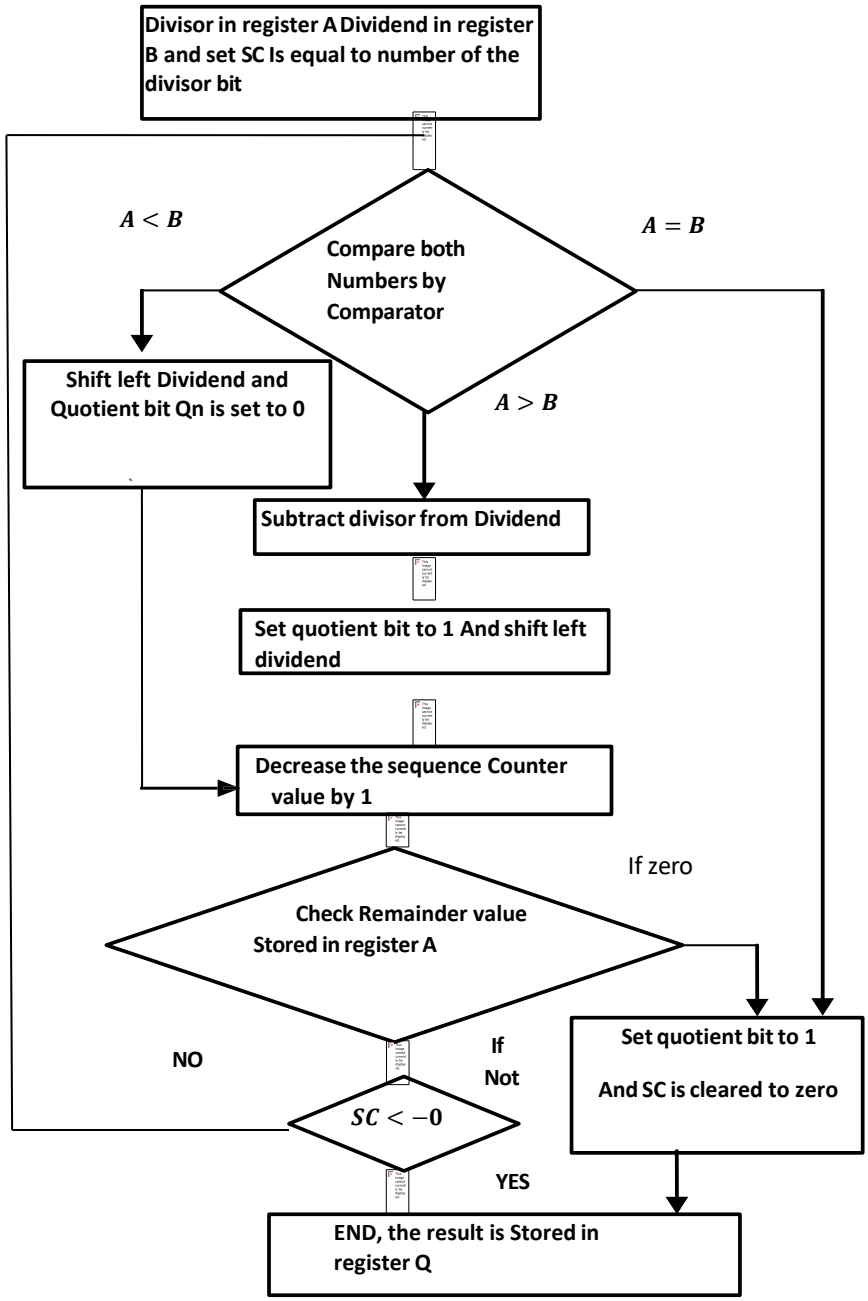

Fig (3) Flow chart for division algorithm by comparison algorithm using magnitude comparator

This flow chart (Fig. 3) delineates the specifications of the binary division operation. This method is straightforward due to the minimal number of instructions. In this procedure, Q represents the quotient register. Initially, we store the dividend in double registers A and Q.

CONCLUSION

This study presents a theoretical framework by introducing an algorithm for binary division utilizing a magnitude comparator and micro-operations. This method improves the integration of hardware and logical implementation, increasing the speed of the binary division algorithm and producing precise quotient bits.

REFERENCES

[1] Arithmetic operations in a binary computer by Robert F.Shaw

[2] An algorithm for non-restoring algorithm by S. Sonycl, Tata Institute of Fundamental Research Bombay, India

[3] Fast 32-bit Division on the DSP56800E Minimized non restoring division algorithm by David Baca

[4] D. Banerji, T. Cheung, and V. Ganesan, "A High-speed Division Method in Residue Arithmetic, "Proceedings of 5th IEEE, Symposium on Computer Arithmetic, Michigan, USA, 1981, pp. 158-164

[5] A Protected Division Algorithm , Published in P. Honey man, Ed., Fifth Smart Card Research and Advanced Application Conference (CARDIS '02), pp. 69–74, Usenix Association, 2002. Marc Joye and Karine Villegas

[6] J. H. Yang, C. C. Chang, and C. Y. Chen, "A High- Speed Division Algorithm in Residue Number System Using Parity- Checking Technique,"

[8] Improved Algorithms for Non-restoring Division and Square Root by Kihwan Jun, B.S.E.E., M.S.E. M.S.E.E

[9] A Division Algorithm Using Bisection Method Residue Number System by Chin-Chen Chang and Jen-Ho Yang

[10] Binary division and square-rooting using Gray code by CK Yuen

[11] VHDL Implementation of Non Restoring Division Algorithm Using High Speed Adder/Sub tractor Sukhmeet Kaur.