DESIGN AND IMPLEMENTATION OF SLOW AND FAST DIVISION ALGORITHM IN COMPUTER ARCHITECTURE

Team name: Solution seeker
Indhu Tharshini N
Department of Computer science and business system
Rajalakshmi Institute of Technology
Chennai , Tamil Nadu , India
indhutharshini.n.2021.csbs@ritchennai.edu.in

Mentor name:Selvakumaran
Department of Artificial intelligence and data science
Rajalaksmi institute of Technology,Chennai,Tamil Nadu,India
selvakumaran.s@ritchennai.edu.in

ABSTRACT

This paper presents a study on the design and implementation of both slow and fast division algorithms in computer architecture . Computer architecture is the backbone of every computing device that plays a major role in our day to day life from personal computers and smartphones to servers and embedded systems. The slow division algorithm, also known as the restoring division algorithm. This method performs division iteratively, one bit at a time. We provide a detailed explanation of the slow division algorithm and discuss its step-by-step execution process and included sample code with output. Furthermore, we developed into fast division algorithm, which aims to reduce the number of iterations required for division by utilizing more complex techniques. Fast division algorithm is also known as non-restoring algorithm. We outline the algorithm's steps and compare its performance characteristics to those of the slow division algorithm. Through this study, we aim to provide a comprehensive understanding of the design and implementation of slow and fast division algorithms in computer architecture. From this research we can help architects and developers to make informed decisions when selecting and optimizing division algorithms, thereby improving the overall efficiency and performance of computer systems.

1.INTRODUCTION

In computer architecture, division operations are fundamental to numerous computational tasks, ranging from basic arithmetic calculations to complex algorithms. Division algorithms play a pivotal role in determining the efficiency and performance of computer systems. Recognizing the significance of division, researchers have developed various techniques to optimize the process and enhance computational speed. Two prominent approaches in this domain are fast and slow division algorithms.

Fast division algorithms are designed to provide high-performance division operations, aiming to reduce the execution time required to obtain accurate results. These algorithms often employ sophisticated mathematical techniques and optimizations to achieve improved efficiency. They are especially crucial in applications that demand rapid division computations, such as real-time signal processing, scientific simulations, and numerical analysis.

On the other hand, slow division algorithms prioritize accuracy and simplicity over speed. They ensure precise division results without employing intricate optimization strategies. Slow division algorithms are commonly used in applications where accuracy is paramount, such as financial calculations, cryptographic *operations, and error-correcting codes. These algorithms ensure that division operations are performed reliably, even at the cost of additional execution time.*

The design and implementation of fast and slow division algorithms involve a thorough understanding of the underlying mathematical principles, hardware constraints, and trade-offs between speed and accuracy. Researchers and computer architects work together to develop division algorithms that align with specific application requirements and hardware capabilities. This paper aims to explore the design and implementation of fast and slow division algorithms in computer architecture. It discusses the theoretical foundations, optimization techniques, and practical considerations involved in achieving fast and accurate division operations. Furthermore, it highlights the impact of hardware advancements on the performance of division algorithms and provides insights into the challenges and future directions of division algorithm design.

By understanding the principles behind fast and slow division algorithms, computer architects can make informed decisions when selecting the most suitable approach for a given application scenario. This research contributes to the advancement of computer architecture and facilitates the development of more efficient and reliable division operations, enabling the execution of a wide range of computational tasks with optimal performance.

2.SLOW DIVISION

In computer architecture, a slow division algorithm is a method for performing division operations that prioritizes accuracy and simplicity over speed. It ensures precise division results without employing complex optimization strategies. Although slow division algorithms may take more time to compute compared to fast division algorithms, they are commonly used in applications where accuracy is crucial.

One of the most well-known slow division algorithms is the restoring division algorithm. Let's explore its workings and illustrate it with a necessary diagram:

Restoring Division Algorithm:

Initialization:

Dividend (D): The number to be divided.

Divisor (d): The number by which the dividend is divided.

Quotient (Q): The result of the division.

Remainder (R): The remaining value after each division step.

Align the Dividend and Divisor:

Determine the bit length of the dividend and divisor.

Align the most significant bit (MSB) of the divisor with the MSB of the dividend.

The remainder initially holds the value of the dividend.

Division Steps:

Compare the divisor with the remainder.

If the divisor is greater or equal to the remainder, subtract the divisor from the remainder.

Set quotient bit to 1.

If the divisor is smaller than the remainder, do not subtract anything.

Set quotient bit to 0.

Shift and Adjust:

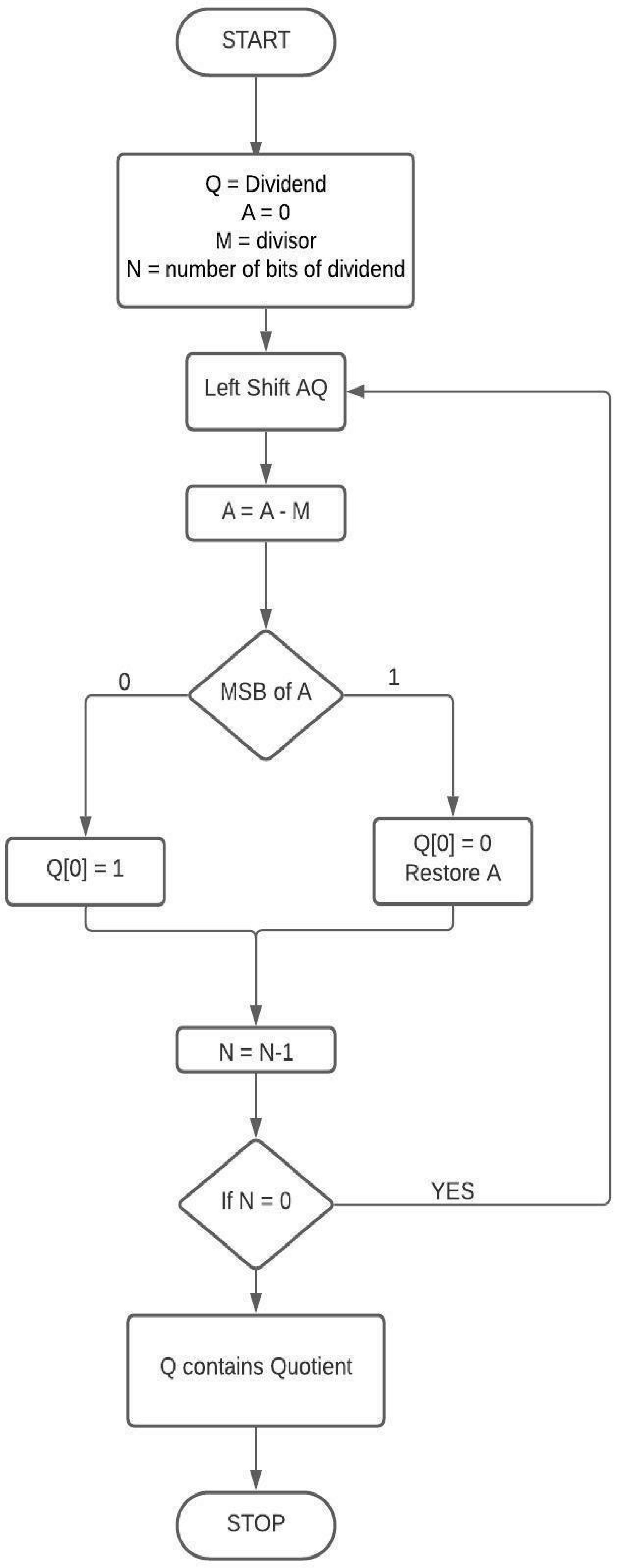
After each comparison, shift the remainder left by one bit.

Bring in the next bit of the dividend to the least significant bit (LSB) of the remainder.

Repeat the division steps until all the dividend bits have been processed.

Termination:

When all the dividend bits have been processed, the quotient is obtained.



The division steps involve comparing the divisor with the remainder at each stage. Depending on the comparison result, the quotient bit is set to 1 or 0. The remainder is then shifted left, and the next bit from the dividend is brought in. This process continues until all dividend bits have been processed.

The slow division algorithm, such as the restoring division algorithm, provides accurate division results by following a straightforward and systematic procedure. However, it may require more computational time compared to faster algorithms. Nonetheless, in applications where accuracy is paramount, slow division algorithms remain an important consideration in computer architecture.

CODE FOR SLOW DIVISION ALGORITHM

```
#include <iostream>

using namespace std;

int main( ){

    // initializing all the variables with Dividend = 9, Divisor = 2.

  int Q = 8,q=1,M=3;

  short N = 4;

  int A = Q;

  M <<= N;

  // loop for division by bit operation.

  for(int i=N-1; i>=0; i--) {

    A = (A << 1)- M;

    // checking MSB of A.

    if(A < 0) {

      q &= ~(1 << i);  // set i-th bit to 0

      A = A + M;

    } else {

      q |= 1 << i;    // set i-th bit to 1

    }

  }
  cout << "Quotient: "<< q;
  return 0;
}
```

OUTPUT:

```
Quotient: 2
```

3.FAST DIVISION

Initialization:

Dividend (D): The number to be divided.

Divisor (d): The number by which the dividend is divided.

Quotient (Q): The result of the division.

Partial Remainder (PR): The intermediate remainder value during the division process.

Align the Dividend and Divisor:

Determine the bit length of the dividend and divisor.

Align the most significant bit (MSB) of the divisor with the MSB of the dividend.

The partial remainder initially holds the value of the dividend.

Division Steps:

Comparing divisor with the remainder:

If the divisor is greater or equal to the partial remainder, subtract the divisor from the partial remainder.

Set quotient bit to 1.

If the divisor is smaller than the remainder, do not subtract anything.

Set quotient bit to 0.

Shift and Adjust:

After each comparison, shift the partial remainder left by one bit.

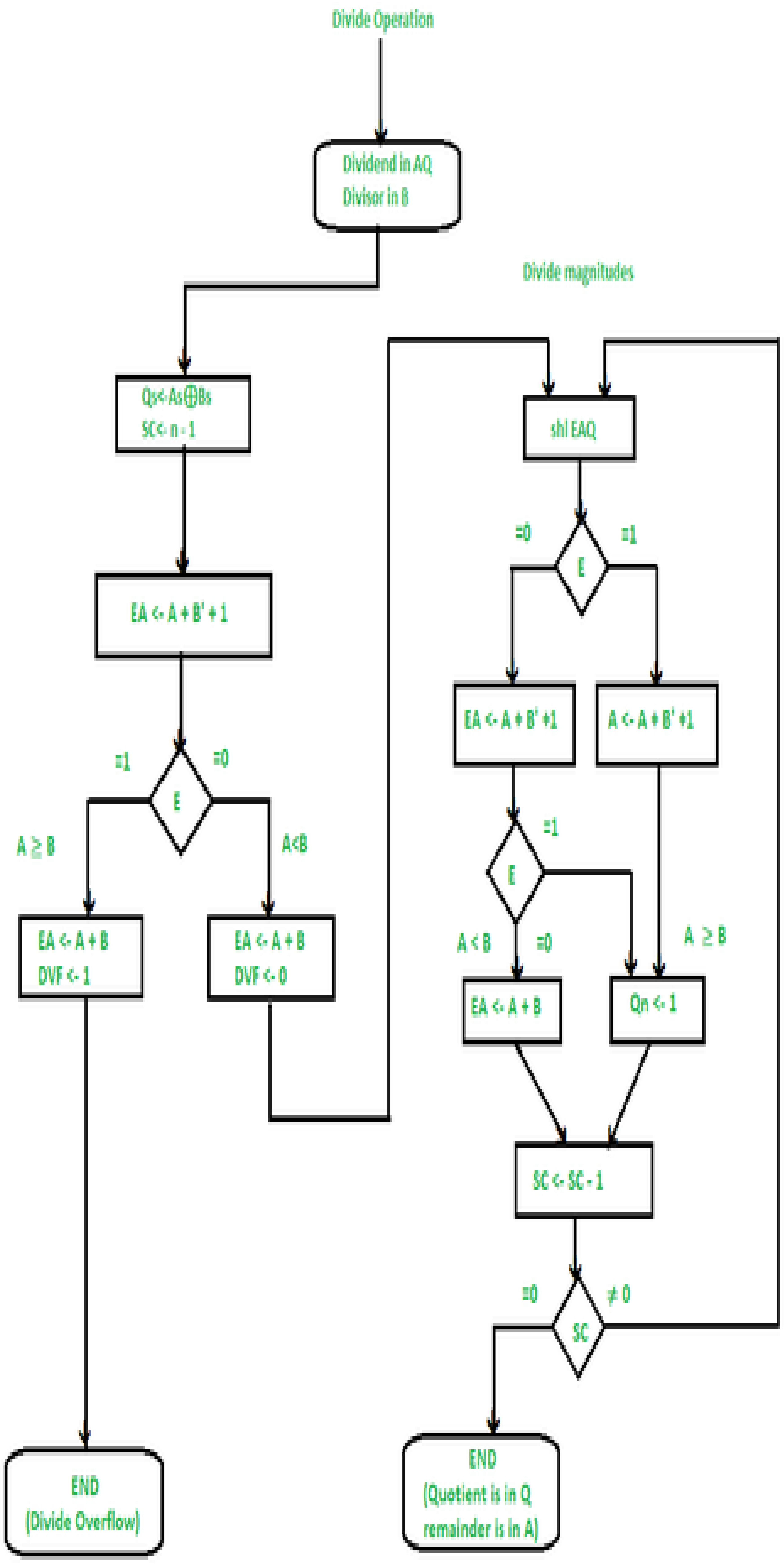
Bring in the next bit of the dividend to the least significant bit (LSB) of the partial remainder.

Repeat the division steps until all the dividend bits have been processed.

Termination:

When all the dividend bits have been processed, the quotient is obtained.The fast division algorithm follows a similar process to the slow division algorithm. It involves aligning the divisor and dividend, comparing the divisor with the partial remainder, shifting the partial remainder, and obtaining the quotient. However, fast division algorithms often include additional optimizations to improve the speed of the division operation. These optimizations can include techniques such as precomputing reciprocal values, using look-up tables, or employing hardware acceleration.In computer architecture, fast division algorithms may utilize parallel processing, pipelining, or specialized hardware units to accelerate the division operation. These optimizations aim to minimize the number of clock cycles required to perform the division and achieve higher throughput.

While a diagram can be helpful in understanding the visual representation of the algorithm, the textual explanation provided above outlines the core steps involved in a fast division algorithm in computer architecture.


Divide Operation
Dividend in AQ
Divisor in B
Qs<-As⊕Bs
SC<- n - 1
EA <- A + B' + 1
=1
=0
E
A ≥ B
A<B
EA <- A + B
DVF <- 1
EA <- A + B
DVF <- 0
END
(Divide Overflow)
Divide magnitudes
shl EAQ
=0
=1
E
EA <- A + B' +1
A <- A + B' +1
=1
E
A < B
=0
A ≥ B
EA <- A + B
Qn <- 1
SC <- SC - 1
=0
≠ 0
SC
END
(Quotient is in Q
remainder is in A)

CODE FOR FAST DIVISION ALGORITHM

```
#include <iostream>
// Fast Division Algorithm
int fastDivision(int dividend, int divisor) {
   int quotient = 0;
   int partialRemainder = dividend;

   for (int i = 31; i >= 0; i--) {
      if ((partialRemainder >> i) >= divisor) {
         partialRemainder -= divisor << i;
         quotient |= 1 << i;
      }
   }

   return quotient;
}

int main() {
   int dividend, divisor;
   std::cout << "Enter dividend: ";
   std::cin >> dividend;
   std::cout << "Enter divisor: ";
   std::cin >> divisor;

      int quotient = fastDivision(dividend, divisor);

     std::cout << "Quotient: " << quotient << std::endl;
   return 0;
}
```

OUTPUT:

```
Enter dividend: 67
Enter divisor: 677
Quotient: 0
```

Conclusion:

we learned about the division algorithm in computer architecture which is the Restoring Algorithm. The proposed division methods proving the correct rounding for division algorithms.

REFERENCE

[1] Foundations of Computer Architecture: Principles and Design., Koffka Khan

[2] Computer System Architecture.., Victor Benevent Raj ALt.Dr.s. Kevin AndrewsLt.,Dr.s.Kevin Andrews

[3] Computer Architecture .,Chitra RaviMohan Kumar S

[4]Division algorithms and implementations, S.F. Obermann; M. J. Flynn

[5]computer arithmetic architectures with redundant

number system, H.R. Srinivas; K.K. Parhi

[6]Fast division algorithm with a small lookup table, P. Hung; H. Fahmy; O. Mencer; M.J. Flynn

[7]Fast division using accurate quotient Approximations to reduce the number of iterations, DC Wong, MJ Flynn - Proceedings 10th IEEE Symposium on ..., 1991 - researchgate.net

[8]An efficient division algorithm and its architecture, SG Chen, CC Li - Proceedings of TENCON'93. IEEE Region 10 ..., 1993 - ieeexplore.ieee.org

[9]Design issues in division and other floating-point operations, SF Oberman, MJ Flynn - IEEE Transactions on computers, 1997 - ieeexplore.ieee.org

[10]A comparative performance analysis for the computer arithmetic based fast division algorithms, Muhammed Erdem Isenkul

[11]An efficient division algorithm and its architecture, *Sau-Gee Chen*, *Chieh-Chih Li*

[12]The implementation of computer modeling methods in the design of nonlinear architecture objects,Siarhei Stsesel