SRT Division Algorithm

Seungho Jang

3/19/2020

Abstract

When it comes to the development of hardware, it is hard to ignore the performance depending on demand of domain. Then what would be the "best proxy" to determine which computer is faster or vice versa as computer architect. One way to observe this proxy is to observe in Arithmetic Logical Unit(ALU). Within these several modules of ALU, how to design algorithms such that it offers fast computation as well as reliable is crucial in these days computing. Such as addition, subtraction, multiplication, and division algorithms have been developed to offer a lot of advantage of data processing and deep learning community. This paper will discuss about SRT division algorithm, how it works, and what benefit provide us better as compared to other algorithm

1 Introduction

As compared to other addition, subtraction, and multiplication algorithm, division algorithm is more complicated because dividing a long number means guessing what number should be in the quotient so that minimum remainder can be obtained. Human were taught and trained how to do divide two numbers in school. But the question would be can computer guess the number correctly or can computer think is remainder proper. Also, as the multiplication algorithm is getting improved, that means there should be a fast and accurate algorithm for division as well. The motivation in this paper is to find/compute the quotient and remainder in given two strings of N(dividend) and D(divisor). Within these division methods in computer architecture world can be divided into two main categories : slow division and fast division. Even within slow division, there are three types of divisions, which they are restoring, non-restoring, and SRT division.Within the category of division algorithm, the simple way to calculate the remainder and quotient is digit recurrence, and most commonly used division algorithm is SRT division, which was taken by the initials of Sweeney, Robertson[2] and Tocher[3]. In this paper, SRT division algorithm will be discussed in-depth, how to implement in software, and observe the time complexity of this algorithm.

2 Body

The division of numbers can be obtained by this mathematical from :

$$\frac{N}{D} = (Q, R)$$

where Q is the quotient and R is the remainder.[4] These operation can be done with human hand easily, but the question is how division method using SRT algorithm can be implemented, what are the input of SRT methods, and what are the assumption. To answer this question, the description below was considered for software implementation.

Input. Two different length of strings: one is for dividend and the other is for divisor. The dividend and divisors are consist of binary fractions such as .CD .E. They can be consist of binary and hexadecimal numbers

Output. Two different length of strings: one is for quotient and the other is for remainder.

Assumption. The dividend is normalized during the division operation, and while this operations, overflow of division can be checked.

When the other division algorithms were considered, their main operation is to subtract. But SRT algorithm uses both addition and subtraction, and the shift operation which is for normalizing the divisor and adjusting the dividend. The basic implementation is to store this divisor and dividend is that half of divisor is stored in each A and Q registers, and the divisor is stored in register B. Within this hardware emphasis, one of the assumption was considered because these register is fixed size, which means when the carry bit of addition and subtraction, overflow could possibly happen. Furthermore, overflow can happen if the high order n bits of dividend register is higher than the divisor. In order to avoid overflow, the value of dividend should be smaller than the divisor. To obtain the smaller value, adjustment can be done by shifting the dividend register to the right as well as half of dividend is smaller than or equal to divisor[1]. To achieve this as implementation wise is to check before the algorithm starts, so that the proper remainder and quotient can be calculated.

2.1 Algorithm

The SRT algorithm consist of three operations as discussed: addition, subtraction, and shift. This algorithm execute until the length of divisor plus one. The detailed of this algorithm is below.

1. Check input whether high order half part of AQ is greater than the divisor. If the value of divisor(A) is higher than the value of B, then shift right until the A is smaller than B, if not, go into step 2.
2. Normalize the B and Get 2's complement of B to do addition.
3. Adjust the AQ by shifting left with same number of position that the divisor was shifted during normalization. When shift happen, next character will be *
4. Subtract A to B, which is the addition of 2's complement of B.
5. After this operations, there will be two cases and loop until all shift is equal to the length of divisor + 1
   (a) If the result of first bit is 1, it is negative result then insert 0 into last significant bit Q(q0), shift the AQ, shift overs 1's, and insert 1's.
   (b) If the result of first bit is 0, it is positive result then insert 1 into last significant bit of Q(q0) and shift over 0's, and insert 0's
6. When the shift is equal to the length of divisor + 1, stop the algorithm.
7. if the most significant bit of A is 1, then shift AQ right to correct the bit, then add normalized B. Then go to step 8
8. the value in the Q is the quotient and the value in the A is the remainder.

By SRT algorithms and the implementation, the remainder and quotient can be obtained. Also, the execution time can be calculated considering each operation is explained below.

Execution Time
1. Each shift for normalizing B takes $3\Delta t$

2. Getting two's complement, each bit of B(length of B) will takes $1\Delta t$
3. Adjusting AQ for each shift take $3\Delta t$
4. Each shift over 1's and 0's take $3\Delta t$
5. Each 2 bit for full adder takes for subtraction and addition $6\Delta t$

3 Analysis

Given the test data, length of bits(strings) versus execution time graph can be obtained as shown Figure 1. Since this graph was generated based on the test data were given, which it was small amount of data, generalization can still be done by observing the graph. Generally, when operand length gets longer and longer, it tends to have longer execution time. As expected, there was positive linear function as shown in the figure 1, but there are some anomaly, meaning that there are some points that are taking short execution time. The reason could be because when there is no shift or less shift for normalizing the divisor(B) as compared to other operands. The reason that some of execution time is longer because when the result string of AQ from subtract operation have positive result, meaning that addition operation is involved or meaning that there are lots of swapping operations between subtract and addition. As the result has shown, the expectation of ratio between execution time and operand length was positive result by observing graph.

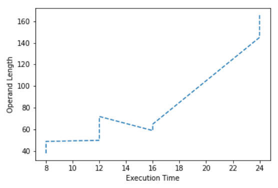

Figure 1: Execution time vs Operand length

4 Challenges

When some of operand length gets larger and larger, the simulator was not working properly to calculate the remainder , quotient and execution time. This can be resolved by fixing bit-wise operation in the simulator because sometime it does not read one by one or it doesn't check if the length of A and length B is same, which that result was not expected. Also, converting hexadecimal to binary was not stable.

Deleting some statement of program was allowed to execute program especially on when the operand bits consist of numbers(not a binary number) such .19. Overall some test data was working properly, but not all data was not feed into simulator.

5 Conclusion

In result, even though there were small amount of data were tested and given, the SRT algorithm simulator was successful to implement for partial data. However, knowing how the algorithm works and string manipulation can fully be beneficial for other division algorithms in the future. Since the execution time over operand length have shown positive linear function, which it satisfied with the expectation.

6 Tools

All code was written in C++ and README.md file was attached for simulator.

References

[1] Hurson Ali. *Division Algorithm, module 2.* 2020.

[2] James E Robertson. "A new class of digital division methods". In: *IRE Transactions on Electronic Computers* 3 (1958), pp. 218–222.

[3] Keith D Tocher. "Techniques of multiplication and division for automatic binary computers". In: *The Quarterly Journal of Mechanics and Applied Mathematics* 11.3 (1958), pp. 364–384.

[4] Wikipedia contributors. *Division algorithm — Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/w/index.php?title=Division_algorithm&oldid=944749477`. [Online; accessed 20-March-2020]. 2020.