Successive Approximation Algorithm for Binary Division of Unsigned Integers

Pritam Bhattacharyya

***Abstract*—This paper presents an efficient binary division algorithm. It is assumed that both the divisor and the dividend are unsigned binary integers. The algorithm employs a successive approximation approach to perform the division. Theoretically, it will take [$n - m + 3$] number of steps to perform a division, where n and m are the number of bits of the dividend and the divisor respectively. The best case of the algorithm occurs when $n \leq m$ and it will then take two steps for its complete execution.**

***Index Terms*—Successive approximation, algorithm, binary division, restoring division, non-restoring division, unsigned integer.**

I. Introduction

Division is one of the four arithmetic operations that we frequently encounter in our daily life. It is of little surprise that, an efficient division algorithm can considerably improve the performance of the ALU component of any digital circuit. In other words, an efficient division algorithm is one of the main aims of a designer while implementing an ALU circuit. The algorithm presented in this paper, solves the problem of binary division, by the use of a successive approximation approach. The algorithm is developed keeping in mind that both the divisor and the dividend are unsigned binary integers.

The algorithm is based on successive approximation of the quotient. The theoretical time taken by the algorithm for division, as proved later, is [$n - m + 3$] steps, where n and m are the number of bits of the dividend and the divisor respectively. The best case of the algorithm occurs when $n \leq m$ and it will then take two steps for completely execution of the division. Here each step corresponds to an iteration of the loop in the algorithm or any extra computations outside the loop, if any.

A brief preliminary about some of the existing integer division algorithms like restoring division algorithm [1]-[4] and non-restoring division algorithm [2]-[4] is presented in the next section, before presenting the successive approximation algorithm. After which, a comparison is made between the different algorithms. The advantages, disadvantages and application scope of the successive approximation algorithm are finally pointed out at the concluding section.

Manuscript received April 25, 2012; revised June 2, 2012.

Pritam Bhattacharyya is with Dept. of Electronics and Communication Engineering, Guru Nanak Institute of Technology, India (e-mail: b.pritam21@gmail.com).

II. Preliminaries

A. Integer Division

The division that is performed by the algorithm presented in this paper is integer division [5], [6], i.e., division of two integers. Let us take two integers A (dividend) and B (divisor) such that, $B \neq 0$ and $A \geq B$, then upon division of A by B, we get two integers Q (quotient) and R (remainder), such that:

$$A = B \times Q + R. \tag{1}$$

Various algorithms exist to perform integer divisions, a few of them are discussed in the subsequent subsections.

B. Restoring Division

Restoring division [1], [2], [3], [4] is a trial-and-error method of approach for binary integer division. The algorithm for restoring division of two integers A (dividend), B (divisor), such that, $B \neq 0$ and $A \geq B$, is as follows [4]:

1) Take an intermediate remainder register R and intermediate quotient register Q, both of size n bits, where n is the number of bits in the dividend.
2) Take:
$$Q = A,\ R = \text{“00..(n bits)..0”}. \tag{2}$$
3) Shift RQ left by one bit.
4) Compute:
$$R = R - M. \tag{3}$$
5) If R is negative, then perform:
$$Q[0] = \text{‘0’},\ R = R + M \text{ (restore)}, \tag{4}$$
else perform:
$$Q[0] = \text{‘1’}. \tag{5}$$
6) Repeat steps ‘iii’ to ‘v’ ($n - 1$) times.
7) The final quotient is in Q and the final remainder is in R.

Here, both the divisor and dividend are taken to be of equal bit length, if there is a mismatch, then the divisor is padded with zeros in the higher order bit positions to make the number of bits equal to the dividend.

C. Non-Restoring Division

The non-restoring algorithm [2]-[4] for binary division of integers A (dividend), B (divisor), such that, $B \neq 0$ and $A \geq B$, solves the problem of the extra restoring step performed by the previous algorithm by the following method:

1) In restoring algorithm, the operation:
$R = R + M$ for $R < 0$ [from Eq. (3) and Eq. (4)],
implies that in the next step, the following operation is

performed:

$$R = 2 \times (R + M) - M, \quad (6)$$

this in turn can be reduced to:

$$R = 2*R + M. \quad (7)$$

2) Since no such operation is performed in case of $R \geq 0$ [Eq. (3) and Eq. (5)], implies that in the next step, the following operation is performed:

$$R = 2*R - M. \quad (8)$$

By replacing the restoring operation [Eq. (4)] with its equivalent operation [Eq. (7)], we arrive at the non-restoring algorithm for binary division of integers, which is as follows [4]:

1) Take an intermediate remainder register R and intermediate quotient register Q, both of size n bits, where n is the number of dividend bits.
2) Take:

$$Q = A, R = \text{“00..(n bits)..0”}. \quad (9)$$

3) If sign of R is ‘0’, then:
   - Left shift RQ by one bit position.
   - Perform:

$$R = R - M, Q[0] = \text{‘0’}. \quad (10)$$

Else if sign of R is ‘1’, then:
   - Left shift RQ by one bit position.
   - Perform:

$$R = R + M, Q[0] = \text{‘1’}. \quad (11)$$

4) Repeat step ‘iii’ $(n - 1)$ times.
5) If sign of R is ‘1’, then:

$$R = R + M. \quad (12)$$

This prevents the need for an extra restoring operation in the restoring step and thus reduces the step computation time. Here also, both the divisor and dividend are taken to be of equal bit length, if there is a mismatch, then the divisor is padded with zeros in the higher order bit positions to make the number of bits equal to the dividend.

III. Successive Approximation Algorithm

In successive approximation approach, we solve the problem by first successively approximating the value of the quotient, according to the integers provided to us, then adjusting it to get the exact value. If A and B is the unsigned dividend and the unsigned divisor respectively, such that $B \neq 0$, and are of n and m bits, respectively, then we can divide them in the following manner:

1) Initialize Q (quotient) of size $(n - m + 1)$ bits, R (remainder) of size m bits and temp (temporary register) of size n bits. All the registers are initialized as zero.
2) If $n \leq m$, then:
   - Compute:

$$\text{temp} = A - B. \quad (13)$$

Take borrow as ‘1’.
   - If, after subtraction, borrow is ‘0’, then:

Q = ‘0’ and $R = A$ (pad higher order bits of A with ‘0’ if necessary, if $n < m$). (14)

Else if, after subtraction, borrow is ‘1’, then:

$$Q = \text{‘1’ and } R = \text{temp}. \quad (15)$$

End of algorithm.

Else if $n > m$, then:
   - Initialize:

p = MSB of temp, q = MSB of A, and r = ‘1’. (16)
   - Initialize down counter:

$$\text{count} = (n - m + 1). \quad (17)$$

   - Initialize:

Q^* (of size $(n - m + 1)$ bits) = $(n - m)$ bits left shifted ‘1’, (18)

B^* (of size n bits) = $(n - m)$ bits left shifted B. (19)

(Successive Approximation Phase:)

If p = q then:

$$\text{temp} = \text{temp}, Q = Q. \quad (20)$$

Else if $p \neq q$ and p = ‘0’, then:

If r = ‘0’, then:

$$\text{temp} = \text{temp} + B^*, \quad (21)$$

$$Q = Q + Q^*. \quad (22)$$

Else if r = ‘1’, then: check the nearest bit position, t, from that of the leading ‘1’ of B^* towards LSB, for which: temp[t] = B^*[t].

If temp[t] = ‘1’, then:

$$r = \text{‘0’}. \quad (23)$$

Else if temp[t] = ‘0’, then:

$$\text{temp} = \text{temp} + B^*, \quad (24)$$

$$Q = Q + Q^*. \quad (25)$$

If no such bit position can be determined then do the operation taking temp[t] = ‘0’.

Else if $p \neq q$ and p = ‘1’, then:

$$\text{temp} = \text{temp} - B^*, \quad (26)$$

$$Q = Q - Q^*. \quad (27)$$

   - For every case except $p \neq q$ and p = ‘0’ and r = ‘1’ and temp[t] = ‘1’, p and q takes the value of the adjacent lower order bit of temp and A respectively.
   - Decrement count by one, right shift Q^* and B^* by one bit position, and repeat steps $d - f$ until count reaches zero.

(Adjustment Phase:)
   - Compute:

$$\text{temp} = A - \text{temp}. \quad (28)$$

Take borrow as ‘1’.

If, after subtraction borrow = ‘1’, then:

$$Q = Q, R = m \text{ lower order bits of temp}. \quad (29)$$

Else if after subtraction, borrow = ‘0’, then:

$$Q = Q - \text{‘1’}, R = m \text{ lower order bits of (temp} + B) \quad (30)$$

End of algorithm.

In the algorithm, by taking borrow = '1', it is implied that we add another higher order bit adjacent to the MSB of the minuend and set it to '1'. The sole purpose of this operation is to prevent the result from becoming negative even if original minuend < subtrahend. After the subtraction, if this bit is found to be '0' then, it implies that original minuend < subtrahend and the original result should have been negative, else if it is found to be '1' then, it implies the opposite. After checking, we discard this higher order bit from further calculations and take the subsequent steps as mentioned in the algorithm.

For example, to divide 1110001_2 (113_{10}) by 10010_2 (18_{10}) we have to perform the following steps:

Let:

$$A = (1110001)_2 \text{ and } B = (10010)_2. \quad (31)$$

Since n (= 7) > m (= 5), we take:

$$Q = (000)_2,\ Q^* = (100)_2,\ \text{temp} = (0000000)_2,\ B^* = (1001000)_2,\ r = (1)_2, \text{ and count} = (3)_{10}. \quad (32)$$

We can then determine that:

$$p = (0)_2,\ q = (1)_2 \text{ and temp}[t] = (0)_2. \quad (33)$$

Detailed calculations for each step while performing the division using successive approximation algorithm, are tabulated in Table I.

TABLE I: VARIOUS STEPS OF THE ALGORITHM WHILE PERFORMING THE DIVISION BETWEEN A = (1110001)2 AND B = (10010)2.

Count	Q^*	Q	B^*
3	100	000 + 100 100	1001000
2	010	100 + 010 110	0100100
1	001	110 + 001 111	0010010
0	xxx	111	xxxxxxx
x	xxx	111 - 1 110	xxxxxxx
temp	p	q	r
0000000 + 1001000 1001000	0	1	1
1001000 + 0100100 1101100	0	1	1
1101100 + 0010010 1111110	0	1	1
11110001 - 1111110 01110011	x	x	x
1110011 + 10010 10000101	x	x	x

temp[t]	Comments
0	$n > m$, thus, successive approximation phase begins. $p \neq q$, p = '0', r = '1' and temp[t] = '0', thus: $Q = Q + Q^*$; temp = temp + B^*; right shift Q^* and B^*. update p and q value. count = count – 1.
0	$p \neq q$, p = '0', r = '1' and temp[t] = '0', thus: $Q = Q + Q^*$; temp = temp + B^*; right shift Q^* and B^*. update p and q value. count = count – 1.
0	$p \neq q$, p = '0', r = '1' and temp[t] = '0', thus: $Q = Q + Q^*$; temp = temp + B^*; right shift Q^* and B^*. update p and q value. count = count – 1.
x	count = 0, thus adjustment phase begins. temp = (A – temp); borrow is taken as '1' (underlined).
x	Since borrow = '0', $Q = Q - 1 = (110)_2$ (underlined); R = 5 lower order bits of (temp + B); i.e., $R = (00101)_2$ (underlined).

Thus, as we can see from above, the division is performed in 7 – 5 + 3 = 5 steps, where, each step corresponds to an iteration of the loop in the algorithm or any extra computations outside the loop, if any. In the first three steps, the algorithm estimated an approximate value of the quotient, which is later adjusted in the last two steps to get its exact value. Thus, the first three steps comprise the successive approximation phase of the algorithm and the last two steps comprise the adjustment phase.

The successive approximation loop runs for ($n - m + 1$) counts, where n is the number of bits of dividend and m is that of the divisor, and the adjustment phase takes two steps. Thus:

The time taken by the algorithm = Time taken by successive approximation phase + time taken by adjustment phase.

(34)

The time taken by the algorithm = ($n - m + 1$) + 2 steps.

(35)

The time taken by the algorithm = ($n - m + 3$) steps.

(36)

However, the best case occurs when $n \leq m$, for which the algorithm evaluates Q and R in two steps. For other cases, the algorithm takes at least four steps, where, each step corresponds to an iteration of the loop in the algorithm or any extra computation outside the loop, if any.

IV. COMPARISON WITH OTHER ALGORITHMS

The time taken by the successive approximation algorithm

is compared in Table II with that of the restoring division algorithm and non-restoring division algorithm in terms of number of steps taken for their respective complete execution. The time required by the other algorithms is obtained from their corresponding algorithm mentioned in the previous section. Here each step corresponds to an iteration of the loop in the algorithm or any extra computations outside the loop, if any.

TABLE II: COMPARISON OF THE NUMBER OF STEPS TAKEN BY SUCCESSIVE APPROXIMATION ALGORITHM WITH RESPECT TO OTHER ALGORITHMS FOR DIVIDENDS OF VARIOUS BIT SIZE.

Number of bits required to represent the dividend, n [bits]	Restoring Method [steps]	Non-Restoring Method [steps]	Successive Approximation Method [steps]		
			Best Case ($m \geq n$)	Minimum for Average Case ($m < n$)	Worst Case ($m = 1$)
4	4	5	2	4	6
5	5	6	2	4	7
6	6	7	2	4	8
7	7	8	2	4	9
8	8	9	2	4	10
16	16	17	2	4	18
32	32	33	2	4	34

V. CONCLUSION

From the previous section, we see that for the best case and the minimum for an average case, number of steps taken for performing the division is considerably lower than the other algorithms as the bit size n increases. Here each step corresponds to an iteration of the loop in the algorithm or any extra computations outside the loop, if any. The time taken for the execution of the worst case is however higher than that of the others, which is a disadvantage of this algorithm. The application scope for this algorithm is in areas where frequent divisions of big unsigned integers are encountered.

ACKNOWLEDGEMENT

The author would like to thank Swagata Bhattacharya, Asst. Prof. of Dept. of Electronics and Communication Engineering, Guru Nanak Institute of Technology, Kolkata, West Bengal, India, for her help and valuable suggestions.

REFERENCES

[1] V. Rajaraman: *Digital Logic and Computer Organization*. PHI Learning Pvt. Ltd., 2006, pp. 206-209.
[2] M. Lu: *Arithmetic and Logic in Computer Systems*. A John Wiley and Sons, Inc. Publication, 2004, pp. 138-143.
[3] D. A. Godse and A. P. Godse: *Digital Design and Computer Organisation*. Technical Publications, 2008, pp. 6-25 – 6-31.
[4] M. Malek. (2002). *Lecture: 12 - ALU (3) – Division Algorithms*, at Humboldt-Universität zu Berlin. [Online]. Available: http://devel-rok.informatik.hu-berlin.de/svn/TI2/2006/folien/pdf/eng_ca12.pdf
[5] K. Hasselström. (2003). *Fast Division of Large Integers A Comparison of Algorithm*. Master's Thesis in Computer Science at the School of Computer Science and Engineering, Royal Institute of Technology, February. [Online]. Available: http://www.treskal.com/kalle/exjobb/original-report.pdf
[6] N. Takagi, S. Kadowaki, and K. Takagi: *A Hardware Algorithm for Integer Division*. Nagoya University, Japan. [Online]. Available: http://arith.polito.it/foils/7_3.pdf