

An Improvement in the Restoring Division Algorithm (Needy Restoring Division Algorithm)

Nitish Aggarwal, Kartik Asooja, Saurabh Shekhar Verma, Sapna Negi
IIIrd Year Students, Department of Computer Engineering
G.B.P.U.A&T, Pantnagar-263145, India

nitish.agg@gmail.com, katrikna@gmail.com, saurabhshekhav@gmail.com, sapna.negil@gmail.com

Abstract— An improvement in the current restoring computer division algorithm which minimizes the number of restorations performed by the computer. Currently, restoration is checked after each shifting of the register but Needy Restoring Division Algorithm checks restoration only when it is needed. The needless restorations are removed by the use of an extra register. This removal decreases the time for the division than taken in the current restoring division algorithm considerably.

Keywords—Restoring Division; Needy Restoration, Logical Shift, Remainder

I. INTRODUCTION

Current restoring division algorithm checks restoration at each iteration after shifting the register.

There are some needless restorations which are not required to be checked:

1. Checking restoration until the first set bit of the dividend is achieved.
2. Checking restoration each time until the value of the register (here, the left half of the Remainder register) equals or becomes greater than the Divisor.

These needless restorations are removed by the use of an extra register. Therefore, three registers have been used here. One is for storing Divisor, second is the Remainder register and third one is M register (the extra register). Divisor register is a 32 bits register. M is also a 32 bits register of which only that bit is on, which is the first set bit of the Divisor from the left, e.g. if **Divisor is 00001010 then M will be 00001000** or if **Divisor is 00111001 then M is 00100000**.

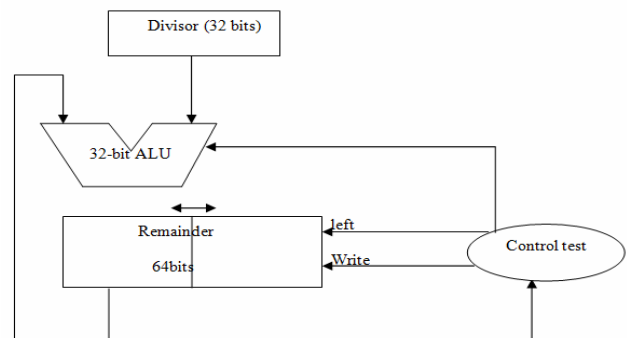


Fig.1 Division Hardware: without M register

The Remainder register is a 64 bits register which initially contains the dividend. The right half of the Remainder register gives the quotient and left half gives the remainder. Here, the Quotient register has been eliminated by shifting the bits of the quotient into the Remainder.

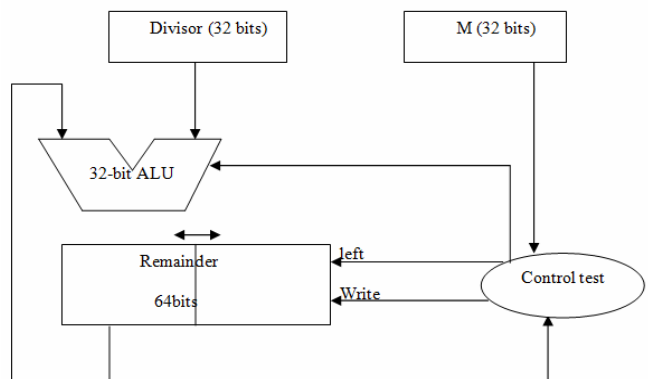


Fig. 2: Division Hardware: with M register

II. NEEDY RESTORATION DIVISION ALGORITHM

The first type of needless restorations (i.e. until the first set bit of dividend is achieved) are left unchecked by first shifting the Remainder register left continuously till the left half of the Remainder register is zero. e.g. Considering an eight bit ALU model, let, if the dividend is **00001001** then Remainder register will be **00000000 00001001** and left half of Remainder register will be 1 or non zero after five times shifting the Remainder, so there is no need of restorations for doing this. It should be done by first shifting the Remainder left continually till the left half is zero and then move further. The second needless restorations are removed by using the register M. It is done by the checking the result for logical AND of Left half of Remainder register and the M register. It is just like that as if we are checking and counting how many bits are required to make the left half of Remainder register equal or greater than the Divisor and then shifting the Remainder register that much times. There is no need of restorations for doing this by using the M register. e.g. Considering an eight bit ALU model, if the Divisor is **00000110** then M is **00000100**. The M register can get its required value while shifting the Remainder register left continuously till the left half of the Remainder register is zero. This can be done independently and in a parallel manner. Suppose Remainder is **00000000 00001001**, then we can continually shift the Remainder register left till (Left half of Remainder & M) is zero. There is no need of restoring the left half of Remainder register during this work. After this the Remainder will be **00000100 10000000**. The left half of the Remainder register is still less than the Divisor, so now the subtraction will be performed and the restoration will be done. But the special point is that only one or nil restoration will be needed in each case after removing the second needless restorations. In this way the current restoring division algorithm is improved as there are less numbers of restorations for doing the division. The efficiency is best improved when *the dividend is very small or divisor is very large*.

Now, here is the flowchart explaining the algorithm steps to be followed for performing the Needy Restoration Division Algorithm.

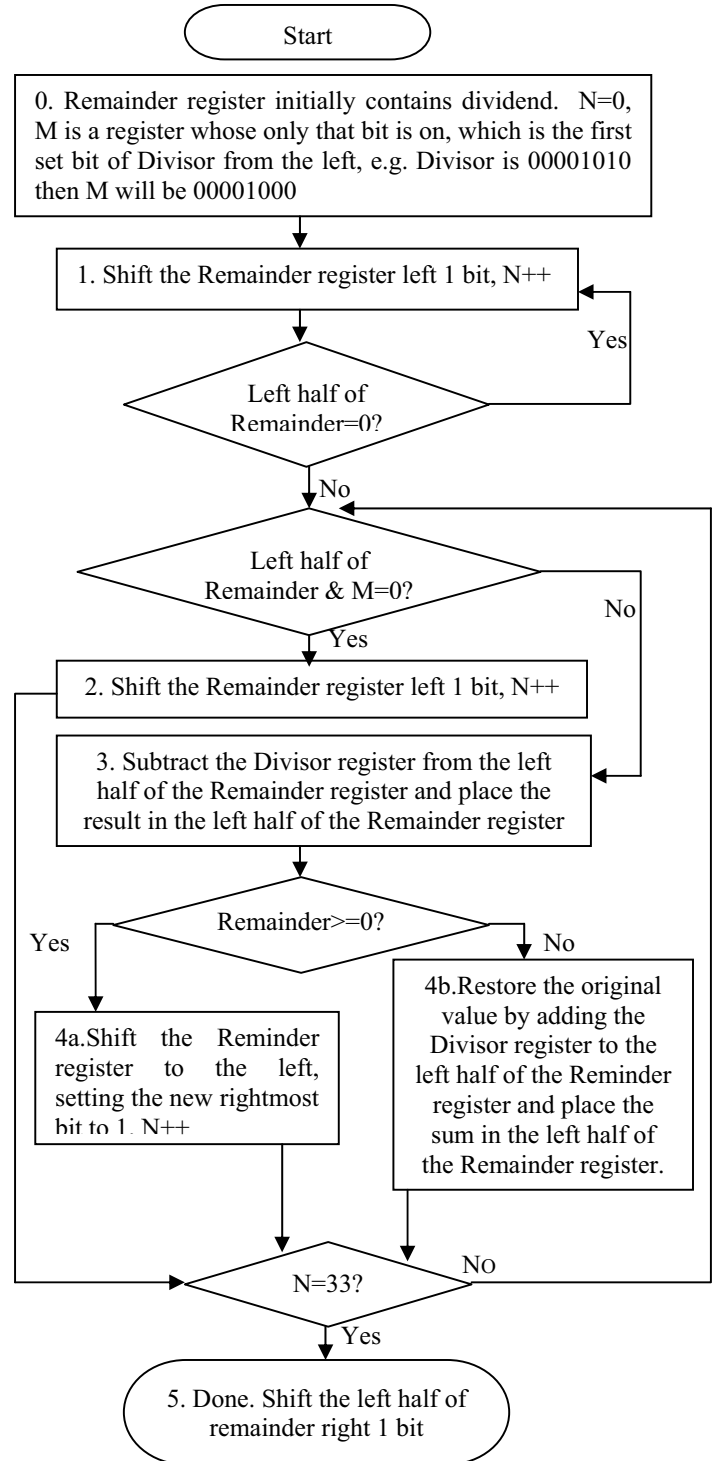


Fig.3: Needy Restoring Division Algorithm

III. EXAMPLE

Considering an eight bits ALU, then according to division hardware shown above, Divisor and M register will be eight bits registers and Remainder register will be 16 bits long.

Let us divide 28 by 9:
Divisor is **00001001**, So, M is **00001000**, Remainder is **00000000 00011100**

Step	Divisor	Remainder	M	N
0: Initial values	00001001	00000000 00111000	00001000	0
1: Shift Rem left 1	00001001	00000000 00111000	00001000	1
1: Left half(Rem)<0 ->Shift Rem left 1	00001001	00000000 01110000	00001000	2
1: Left half(Rem)<0 -> Shift Rem left 1	00001001	00000000 11100000	00001000	3
1: Left half(Rem)<0 -> Shift Rem left 1	00001001	00000001 11000000	00001000	4
2: Left half(Rem)>0, Left half(Rem)&M=0 -> Shift Rem left 1	00001001	00000011 10000000	00001000	5
2: N<9 and Left half (Rem) & M=0 -> Shift Rem left 1	00001001	00000111 00000000	00001000	6
2: N<9 and Left half (Rem) & M=0 -> Shift Rem left 1	00001001	00001110 00000000	00001000	7
3: Left half (Rem) & M = 1 -> Left half(Rem)=Left half(Rem)-Div	00001001	00000101 00000000	00001000	7
4a: Rem>=0 -> Shift Rem left 1 and R0=1	00001001	00001010 00000001	00001000	8
3: Left half (Rem) & M = 1 -> Left half (Rem) =Left half (Rem)-Div	00001001	00000001 00000001	00001000	8
4a:Rem>=0 ->Shift Rem left 1 and R0=1	00001001	00000010 00000011	00001000	9
5. N=9->Shift left half (Rem) right 1	00001001	00000001 00000011	00001000	9

Fig. 4: Example: Dividing 28 by 9

IV. FAVOURABLE CASES

Small dividend *or* large divisor will cause many needless restorations in the restoring division algorithm. But this algorithm (Needy Restoring Division Algorithm) will make the restoring division more efficient by cutting off those restorations as explained above. The best favourable cases will occur when there are occurrences of both the small dividend and the large divisor.

Considering an eight bit ALU:

The Best "One"

This condition occurs when the Dividend is 1 (i.e. **00000001**) and the most significant bit (MSB) of the Divisor is 1 (i.e. **10001010** or **10111000** etc.). In this maximum restoration occurs in case of current restoring division algorithm but the Needy division algorithm will greatly improve the efficiency by cutting off all those restorations.

V. UNFAVOURABLE CASES

Large dividend *or* small divisor will cause very less number of needless restorations in the current restoring division algorithm. So the Needy division algorithm will not create much difference where both the cases occur i.e. large dividend and small divisor also, but even then it will be more efficient than the restoring division algorithm. In any case it will be better than the current restoring division algorithm.

The Worst "One"

This condition occurs when the Divisor is 00000001 and the MSB of the Dividend is 1 (i.e. **10001010** or **10111000** etc.). In this case there will be no improvement in the efficiency of the current restoring division algorithm.

Finally, we can say that in every case Needy division algorithm will improve the efficiency of the current restoring division algorithm.

VI. CONCLUSION

Needy Restoration Division Algorithm provides an improvement over the current restoring division algorithm by removing out the needless restorations, thereby it takes less time for the division. Moreover, in the above mentioned

favourable cases, the difference between the time taken by the current and needy algorithm is considerable.

VII. REFERENCES

- [1] Patterson, D.A and Hennessy, J.L., *Computer Organization and Design: The Hardware/Software Interface*.1993 San Mateo, CA: Morgan Kaufmann Publishers, Second Edition, 1998.
- [2] Stallings, W, *Computer Organization and Architecture (Designing for Performance)*, Upper Saddle River, New Jersey: Prentice Hall Publishers, Fifth Edition, 1999.
- [3] Hennessy, J.L. and Patterson, D.A., *Computer Architecture: A Quantitative Approach*, 1990. Morgan Kaufmann Publishers, Inc. San Mateo, CA. Second Edition, 1995.
- [4]http://www.ee.ucla.edu/~ingrid/ee213a/lectures/division_presentation V2.pdf