A Hybrid Decimal Division Algorithm

Soonyoul Kwon * Jonghwa Choi * Jinsub Park * Seonkyoung Han * and Younggap You *
Dept of Information and Communication Engineering, Chungbuk National University,
Korea, 361-763
Tel : +81-43-271-2480 Fax : +81-43-271-2480 E-mail: sykwon@hbt.chungbuk.ac.kr

Abstract:
This paper presents a hybrid decimal division algorithm to improve division speed. In a binary number system, non-restoring algorithm has a smaller number of operations than restoring algorithm. In decimal number system, however, the number of operations differs with respect to quotient values. Since one digit ranges 0 to 9 in decimal, the proposed hybrid algorithm employ either non-restoring or restoring algorithm on each digit to reduce iterative operations. The selection of the algorithm is based on the remainder values. The proposed algorithm improves computation speed substantially over conventional algorithms by decreasing the number of operations.

Keywords: decimal division, BCD, restoring, non-restoring

1. INTRODUCTION

Decimal computation has not been popular due to the computation speed and hardware cost. The bit-efficiency of a decimal system is relatively lower than that of the binary encoding scheme [1]. Binary coded decimal (BCD) is a typical encoding scheme where four bits represent one decimal digit. Division, on the other hand, is one of the slowest arithmetic operation; SRT division is the fastest one for binary division [3,4]. Decimal division has not been well studied so far.

Division in a decimal system becomes increasingly important due to fractional conversion errors to and from the binary number system. Conversion of a decimal number into its corresponding binary number brings truncation error and thus computation errors become inherent. Even computation using floating point numbers is not free from this type of conversion error [1,5,6]. Decimal computation does not suffer from this conversion error problem.

This paper proposes a new division algorithm based on decimal numbers with efficient computation sequences reducing the number of additions or subtractions. The algorithm is to select restoring or non-restoring steps based on the relative magnitudes of partial remainders. Section 2 describes the conventional division algorithms for decimal computation, Section 3 explains the proposed algorithm, Section 4 evaluates the proposed algorithm, and Section 5 draws conclusion.

2. CONVENTIONAL DECIMAL NUMBER DIVISION

Decimal division is an extension of binary division. Repeated subtractions or additions based on the signs of partial remainders form the main body of the computation. Restoring algorithm iterates the subtraction and shift operations. The sign of the partial remainder should be positive after the generation of the current quotient digit. Trial subtraction of a divisor from a partial remainder yields a sign for restoring operations. In a decimal division, trial subtractions of divisor continue until the sign changes and then the addition to restore the sign follows. Figure 1 presents a pseudo-code of a restoring decimal division algorithm and Figure 2 illustrates its flow diagram.

```
p_-1 = X
q_-1 = 0
for i=0 to m-1
        if p_i > 0 then
                p_i = 2^4 p_i-1 - D;
                q_i = q_i-1 + 1;
        else
                p_i = 2^4 p_i-1 + D;
                q_i = q_i-1 - 1;
                i++;
return (p,q);
```

Figure 1. Pseudo-code for a restoring decimal division algorithm

The algorithm begins with an input of initial values of a dividend and a divisor. The dividend itself is the initial partial remainder. In the process of a division operation, quotient value increases as the divisor subtracted from the partial remainder until the sign of the resultant partial remainder changes. Final restoring performs an addition of the divisor to the partial remainder and a decrement of the quotient value by 1. This process continues after one digit shift of the partial remainder until all the effective digits are covered.

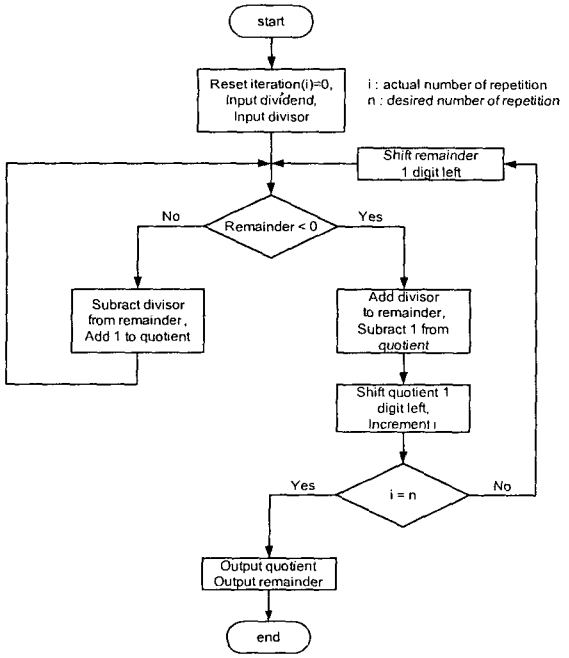

Figure 2. Restoring decimal division algorithm

Non-restoring division performs subtractions and additions alternatively along the digit shifts. It performs subtractions (addition) of the divisor for a positive (negative) partial remainder. After the sign of the partial remainder changes, a digit shift follows without a sign restoration. Figure 3 and Figure 4 show its pseudo-code and algorithm flow, respectively.

```
p_-1 = X
q_-1 = 0
p_0 = 2^4 p_-1 - D
for i=0 to m-1
        if no_change then
                if p_i > 0 then
                        p_i = p_i - D;
                        q_i = q_i + 1;
                else
                        p_i = p_i + D;
                        q_i = q_i - 1;
        else
                p_i = 2^4 p_i-1;
                q_i = 2^4 q_i-1 + 1;
                i++;
return (p,q);
```

Figure 3. Pseudo code of a non-restoring algorithm

The initial value of a dividend is the initial partial remainder. In the process of a division operation, quotient value increases by one as the divisor subtracted from the partial remainder as long as the sign of the resultant partial remainder is positive. Then one digit shift follows. When the sign of the partial remainder is negative, additions of the divisor continues until the remainder sign changes. Each addition accompanies decrease of quotient values by 1. This process continues until all the effective digits are covered. We now turn to the proposed new algorithm saving addition or subtraction operations and thereby speeding up division.

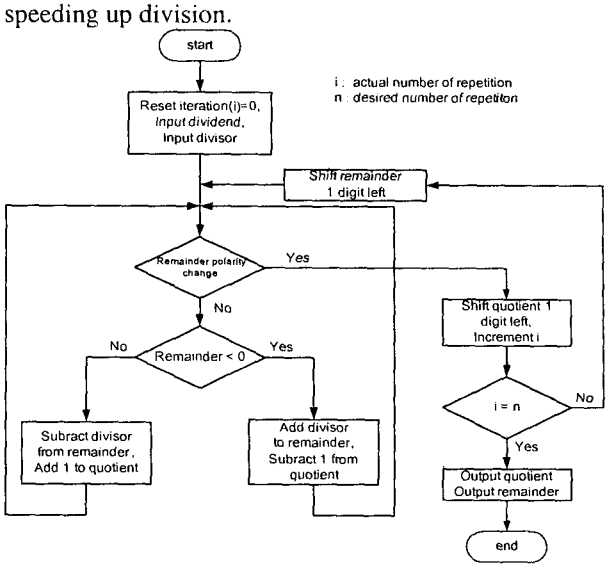

Figure 4. Non-restoring division algorithm

3. THE PROPOSED HYBRID DECIMAL DIVISION

The foregoing restoring and non-restoring algorithms can be combined to produce a faster hybrid algorithm. This section describes the detailed algorithm and its implementation.

3.1. The Proposed Hybrid Division

The number of additions or subtractions of division algorithms differs by n bits maximum for binary numbers. Operations on decimal numbers make this difference even larger due to the ranges of digit values of 0 through 9. Simple application of conventional division algorithm does not guarantee the minimum number of operations. Selective application of restoring and non-restoring algorithm may bring a smaller number of operations.

The proposed hybrid algorithm reduces the number of operations selecting division algorithm independently on each digit. To determine a quotient digit, it selects the non-restoring (restoring) division algorithm when the previous partial remainder is greater (less) than the current partial remainder. The pseudo-code and the flow of the proposed hybrid algorithm are shown in Figure 5 and Figure 6, respectively.

The value of a dividend is the initial partial remainder. In the process of this hybrid division operation, the quotient value of a digit is determined along the conventional restoring or non-restoring algorithm selected for the digit. When the non-restoring division algorithm is selected, the corresponding quotient value increases by one as the divisor subtracted from the partial remainder as long as the sign of the resultant partial remainder is positive. Then one digit shift follows. When the sign of the partial remainder is negative, additions of the divisor continues until the remainder sign changes. Each

addition accompanies decrease of quotient values by 1. When the restoring division is selected, the quotient value increases as the divisor subtracted from the partial remainder until the sign of the resultant partial remainder changes. Final restoring performs an addition of the divisor to the partial remainder and a decrement of the quotient value by 1.

```
p_-1 = X
q_-1 = 0
p_0 = 2^4 p_-1 - D
for i=0 to m-1
  if not_change then
        if p_i > 0 then
              p_i = p_i - D;
              q_i = q_i + 1;
        else
              p_i = p_i + D;
              q_i = q_i - 1;
  else
        if p_i-1 > p_i then
              p_i = 2^4 p_i-1;
              qi = 2^4 q_i-1 + 1;
        else
              if p_i > 0 then
                    p_i = p_i - D;
                    q_i = q_i + 1;
              else
                    p_i = p_i + D;
                    q_i = q_i - 1;
                    p_i+1 = 2^4 p_i;

                    q_i+1 = 2^4 q_i +1;
                    i++;
return (p, q);
```

Figure 5. Pseudo code of the proposed hybrid algorithm

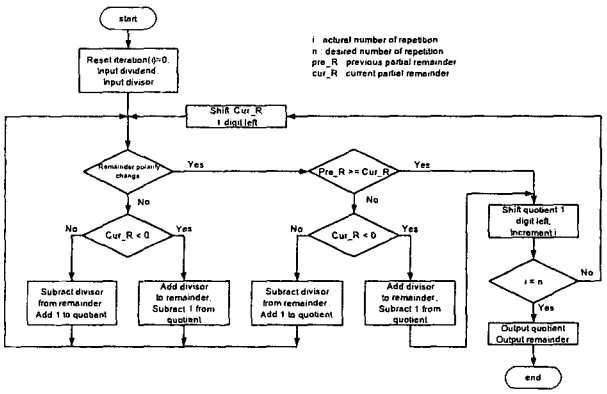

Figure 6. Proposed hybrid division algorithm

3.2. Addition/Subtraction Circuit Design for Hybrid Division

Addition and subtraction with excess-3 code is faster than conventional BCD encoding due to the 9-s complement nature of the decimal arithmetic [6]. In the hybrid division algorithm, each digit position requires multiple addition or subtraction operations performed to meet the selected division algorithm for the digit position. The excess-3 code is not a weighted number system but yields faster addition and subtraction. The addition and subtraction circuit design is illustrated in Figure 7. It comprises a BCD carry lookahead adder and BCD to excess-3 converters.

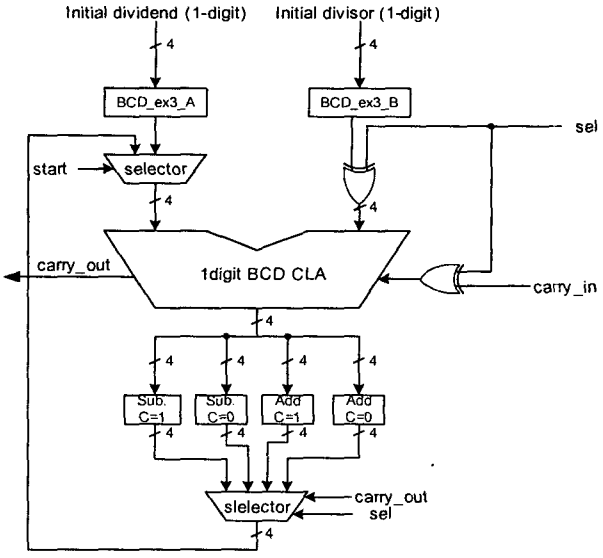

Figure 7. One-digit decimal addition/subtraction block

The signal *sel* is to select either addition *(sel=0)* or subtraction *(sel=1)*. For an addition, the original augend is used as input. For a subtraction, the augend is converted into the 9's complement form. Delay due to the carry (borrow) propagation is reduced employing the excess-3 code and BCD carry lookahead adder [5]. It needs conversion to and from the weighted BCD numbers.

Conversion can be classified into six cases depending on the carry and borrow generation at a digit position. There are two cases for addition and four cases for subtraction. When an addition generates no carry, the conversion subtracts three from an excess-3 coded number. When it finds a carry, it adds three. The equation (1) describes this conversion process for these two cases of additions.

$$\begin{aligned} Y_i &= X_i - 3 \quad \text{if } carry_i = 0 \\ Y_i &= X_i + 3 \quad \text{if } carry_i = 1 \end{aligned} \tag{1}$$

When no borrows involved both in current and previous subtractions it subtracts three from excess-3 coded numbers. In the case of no borrow at the current digit position and a borrow from the previous digit position, it subtracts two from excess-3 coded numbers. With a borrow generation at the current position and no previous borrow, it adds three. Finally it adds four for the case of borrow generations both at current and at previous digit positions. Equation (2) describes these four cases of subtractions.

$$\begin{aligned} Y_i &= X_i - 3 \quad \text{if } carry_i = 0 \text{ and } carry_{i-1} = 0 \\ Y_i &= X_i - 2 \quad \text{if } carry_i = 0 \text{ and } carry_{i-1} = 1 \\ Y_i &= X_i + 3 \quad \text{if } carry_i = 1 \text{ and } carry_{i-1} = 0 \\ Y_i &= X_i + 4 \quad \text{if } carry_i = 1 \text{ and } carry_{i-1} = 1 \end{aligned} \tag{2}$$

4. VERIFICATION AND EVALUATION

The non-restoring algorithm needs ten operations increase per digit for the worst case and 5.5 operations increase per digit in average, respectively [3]. The maximum number of increase of additions or subtractions to process one divisor digit is eleven and the average number is six for the conventional restoring division algorithm. The proposed hybrid algorithm requires six operations increase per digit for the worst case and 3.5 operations increase per digit in average. For a single digit number, all the three algorithms require 11operations for the worst case. All the three algorithms perform subtractions until quotient digits are settled, and final adjustment to get a correct sign. For a larger number of digits the number of operations increases proportional to the number of digits.

For a two digit quotient case, restoring algorithm performs 22 operations, whereas the proposed hybrid algorithm performs 17 operations. For three digit quotients, restoring algorithm demands 33 operations, non-restoring algorithm carries out 29 operations, and the proposed algorithm performs 23 operations. Figure 8 and table 1 illustrate the increase of operations per digit for the three algorithms for comparison purpose.

The number of increases of the three algorithms has been calculated using a C program running in a Pentium 4 processor with 512Mbytes of main memory. Dividends checked range from 1 through 9,999,999 and divisors ranges from 1 through the value of the selected dividend. The program runs for a full day to check those ranges of numbers and their results take 4Gbytes of storage.

Regularity in the increase of operations has been observed from all the three algorithms. For an n-bit quotient, restoring algorithm requires 11n addition/subtraction and 10n for restoring algorithm yielding even number of quotient digits. For n=4i-3 (i=1,2,3...) the number is 10n+1, and for n=4i+1 the number is 10n-1.

For the restoring algorithm, the worst case yields a quotient digit value of 9. Thus the maximum number of operations is for the quotient values of 999..., and the minimum case is for the quotient values of 1000.... For the non-restoring algorithm, the worst case yields the quotient digit values of 909090... , and the quotient values of 19090... are obtained with the minimum number of operations. The proposed hybrid algorithm requires the maximum number operations to produce quotient values of 94444... , and the quotient values of 190190... is the quotient value of the best cases. These observations shorten the computer evaluation time substantially.

5. CONCLUSION

Decimal computation is human friendly and eliminates conversion errors which are critical for some monetary accounting. Conventional decimal division algorithms, restoring and non-restoring division, follow their binary predecessors and are found inefficient in computation. The proposed hybrid division algorithm selects either restoring or non-restoring division on each digit yielding a smaller number of additions or subtractions than conventional ones. For a 64digit quotient, the proposed algorithm requires 1.8 times and 1.6 times less number of operations than the restoring and non-restoring algorithm, respectively. Excess-3 code based addition and subtraction further reduces the operations speed.

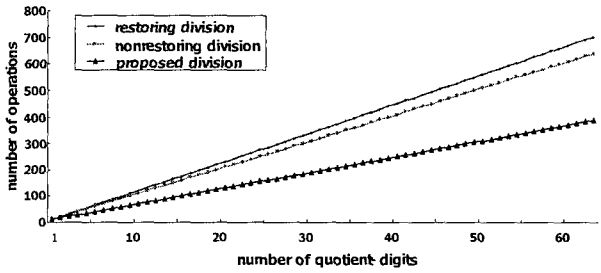

Figure 8. Number of operations of division algorithm

Table 1. The maximum number of add/subtraction operations of each algorithm

Number of quotient	Restoring division	Non-restoring division	The proposed Hybrid
1	11	11	11
2	22	20	17
3	33	29	23
4	44	40	29
5	55	51	35
6	66	60	41
7	77	69	47
8	88	80	53
9	99	91	59
10	110	100	65
16	176	160	101
32	352	320	197
64	704	640	389

References

[1] M. F. Cowlish, et al., "A decimal floating-point specification," *Proc. 15th IEEE Symposium on Computer Arithmetic*, pp. 147-154, June 2001.

[2] M. Cowlishaw, "Densely packed decimal encoding," *IEE Proc. Comput. Digit. Tech.*, vol. 149, no. 3, pp. 102-104, May. 2002.

[3] S. Hermann, *Decimal Computation*, Wiley Inter-science Publication, 1974.

[4] P. Behrooz, *Computer Arithmetic*, Oxford University Press, 2000.

[5] J. Choi and Y. You, "Carry lookahead design for high speed decimal addition," *IEEK J.*, vol. 40CI, no. 5, pp. 241-249, Sept. 2003.

[6] J. Choi, S. K. Han and Y. You, "Excess-3 coded decimal adder/subtracter design," *IEEK J.*, vol. 40CI, no. 6, pp. 348-354, Nov. 2003.

[7] M. S. Schmookler and A. Weinberger, "High speed decimal addition," *IEEE Transactions on Computers*, vol. C-20, no. 8, pp. 862-866, August 1971.

[8] K. Israel, *Computer Arithmetic Algorithms*, 2nd Ed, A K Peters, Ltd. 2002.