US005784307A

United States Patent [19]

Sheaffer

[11] Patent Number: 5,784,307

[45] Date of Patent: Jul. 21, 1998

[54] **DIVISION ALGORITHM FOR FLOATING POINT OR INTEGER NUMBERS**

[75] Inventor: **Gad S. Sheaffer**, Haifa, Israel

[73] Assignee: **Intel Corporation**, Santa Clara, Calif.

[21] Appl. No.: **657,779**

[22] Filed: **May 31, 1996**

[51] **Int. Cl.**[6] **G06F 7/52**

[52] **U.S. Cl.** **364/766**; 364/767

[58] **Field of Search** 364/752, 761, 364/762, 764, 766, 767

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,939,686	7/1990	Fandrianto	364/752
5,357,455	10/1994	Sharangpani et al.	364/767
5,377,135	12/1994	Kuroiwa	364/767
5,386,376	1/1995	Girard et al.	364/767

OTHER PUBLICATIONS

"Computer Architecture A Quantitative Approach, Second Edition", David A. Patterson and John L. Hennessy, Morgan Kaufmann Publishers, Inc., 1996. p. A1–A73.

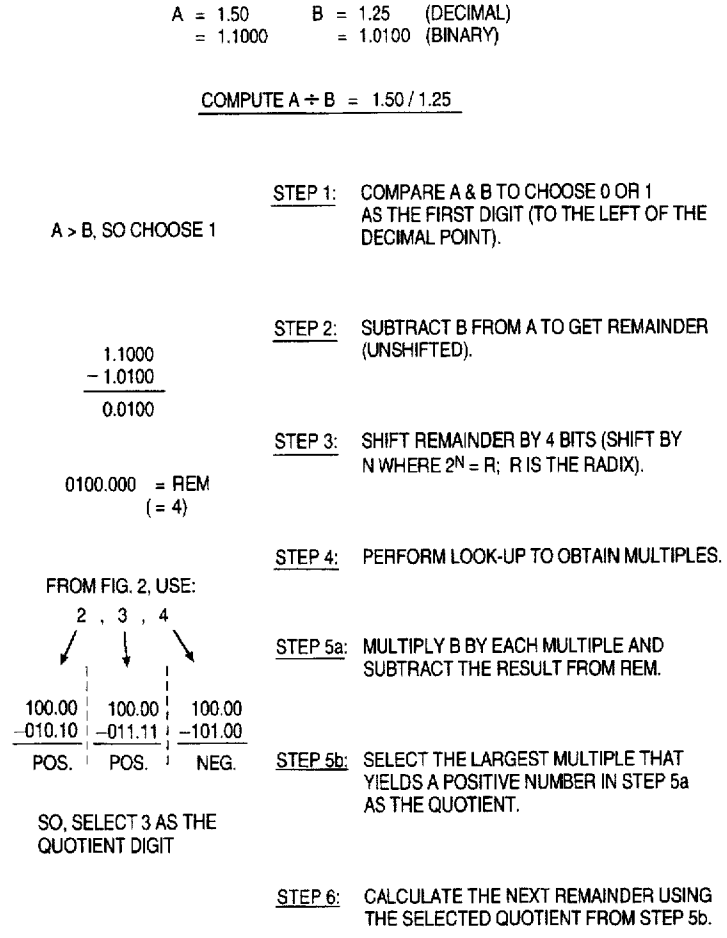
Primary Examiner—Chuong Dinh Ngo

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A computer-implemented algorithm for dividing numbers involves subtracting the divisor from the divided to generate a first intermediate result, which is then shifted by N-bits to obtain a remainder value. A portion of the remainder and a portion of the divisor are utilized to generate one or more multiples from a look-up table, each of which is multiplied by the divisor to generate corresponding second intermediate results. The second intermediate results are subtracted from the remainder to generate corresponding third intermediate results. The largest multiple which corresponds to a third intermediate result having a smallest positive value is the quotient digit. The third intermediate result that corresponds to the largest multiple is the remainder for the next iteration.

37 Claims, 7 Drawing Sheets



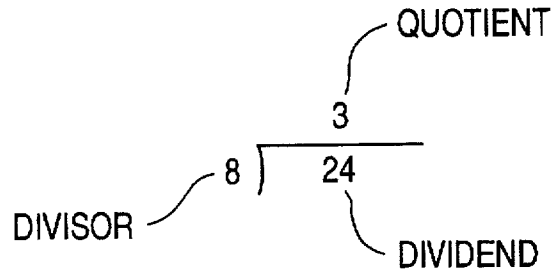


FIG_1A

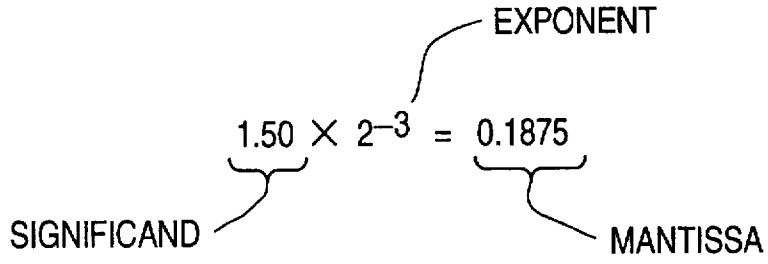


FIG_1B

$$(S_1 \times 2^{e_1}) / (S_2 \times 2^{e_2}) =$$

$$\underbrace{(S_1 / S_2)}_{\text{QUOTIENT}} \times 2^{(e_1 - e_2)}$$

FIG_1C

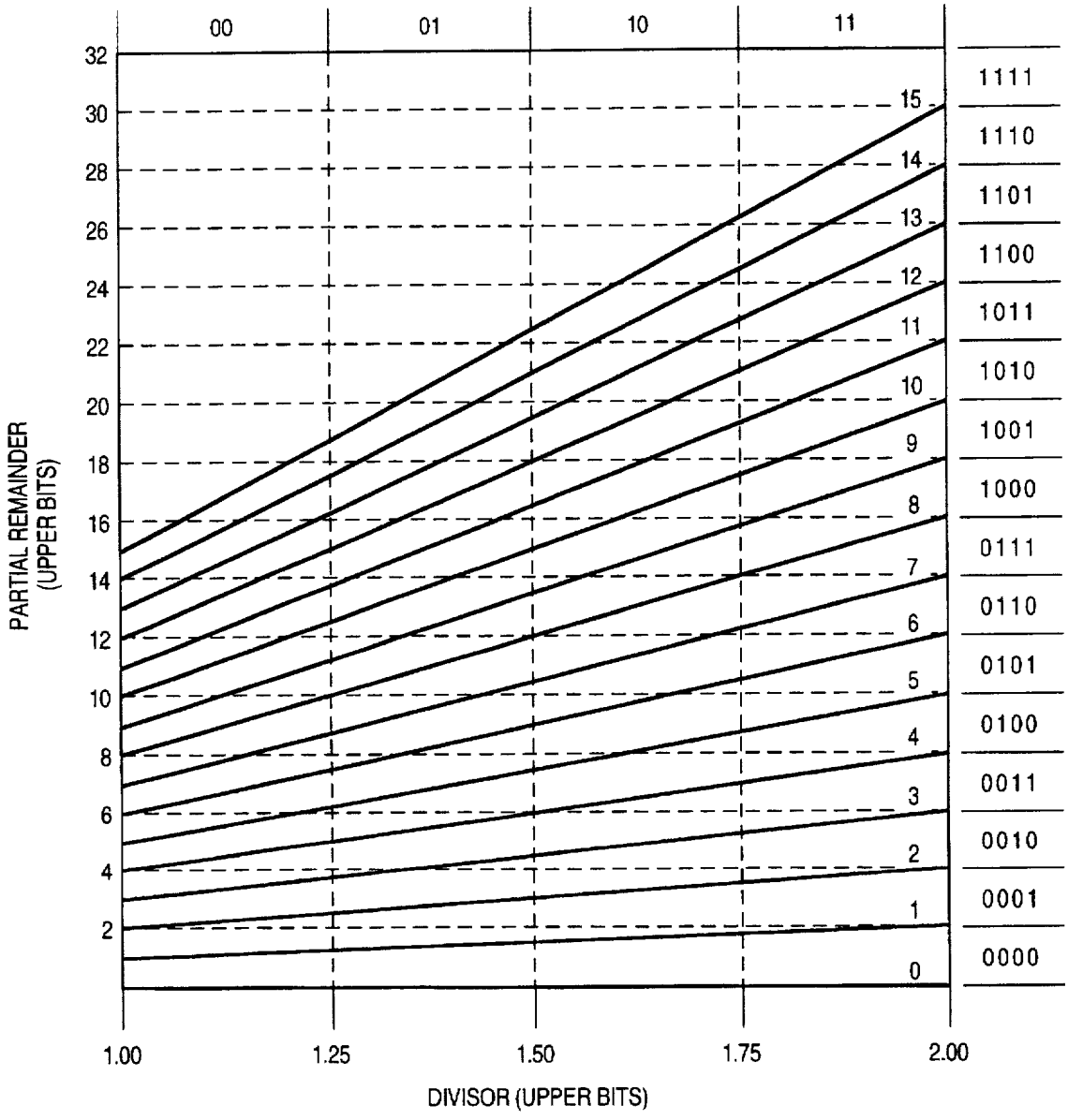

00
01
10
11
32
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
1111
1110
1101
1100
1011
1010
1001
1000
0111
0110
0101
0100
0011
0010
0001
0000
PARTIAL REMAINDER (UPPER BITS)
1.00
1.25
1.50
1.75
2.00
DIVISOR (UPPER BITS)

FIG_2

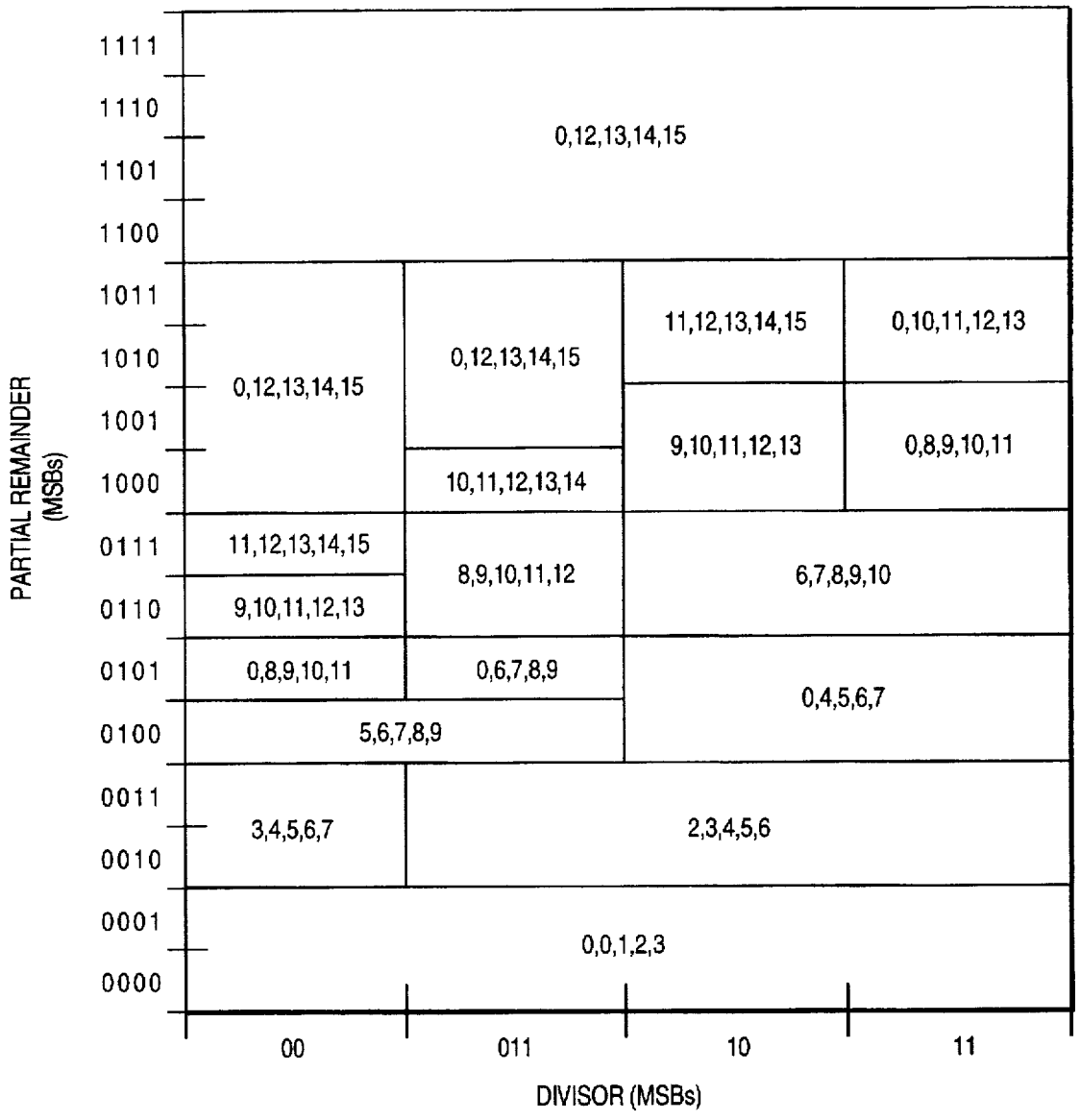

PARTIAL REMAINDER (MSBs)
1111
1110
1101
1100
1011
1010
1001
1000
0111
0110
0101
0100
0011
0010
0001
0000
0,12,13,14,15
0,12,13,14,15
0,12,13,14,15
10,11,12,13,14
11,12,13,14,15
0,10,11,12,13
9,10,11,12,13
0,8,9,10,11
11,12,13,14,15
9,10,11,12,13
8,9,10,11,12
6,7,8,9,10
0,8,9,10,11
0,6,7,8,9
0,4,5,6,7
5,6,7,8,9
3,4,5,6,7
2,3,4,5,6
0,0,1,2,3
00
011
10
11
DIVISOR (MSBs)

FIG_3

MULTIPLE	FIRST SUMMAND	SECOND SUMMAND	THIRD SUMMAND
0	0	0	0
1	0	0	1
2	0	2	0
3	0	2	1
4	4	0	0
5	4	0	1
6	4	2	0
7	4	2	1
8	8	0	0
9	8	0	1
10	8	2	0
11	8	2	1
12	8	4	0
13	8	4	1
14	16	$\overline{2}$	'1
15	16	'1	$\overline{1}$

FIG_4

A = 1.50 B = 1.25 (DECIMAL)
= 1.1000 = 1.0100 (BINARY)

COMPUTE A ÷ B = 1.50 / 1.25

STEP 1: COMPARE A & B TO CHOOSE 0 OR 1 AS THE FIRST DIGIT (TO THE LEFT OF THE DECIMAL POINT).

A > B, SO CHOOSE 1

STEP 2: SUBTRACT B FROM A TO GET REMAINDER (UNSHIFTED).

$$\begin{array}{r} 1.1000 \\ -\ 1.0100 \\ \hline 0.0100 \end{array}$$

STEP 3: SHIFT REMAINDER BY 4 BITS (SHIFT BY N WHERE $2^N = R$; R IS THE RADIX).

0100.000 = REM
(= 4)

STEP 4: PERFORM LOOK-UP TO OBTAIN MULTIPLES.

FROM FIG. 2, USE:

2 , 3 , 4

STEP 5a: MULTIPLY B BY EACH MULTIPLE AND SUBTRACT THE RESULT FROM REM.

100.00	100.00	100.00
−010.10	−011.11	−101.00
POS.	POS.	NEG.

STEP 5b: SELECT THE LARGEST MULTIPLE THAT YIELDS A POSITIVE NUMBER IN STEP 5a AS THE QUOTIENT.

SO, SELECT 3 AS THE QUOTIENT DIGIT

STEP 6: CALCULATE THE NEXT REMAINDER USING THE SELECTED QUOTIENT FROM STEP 5b.

FIG_5

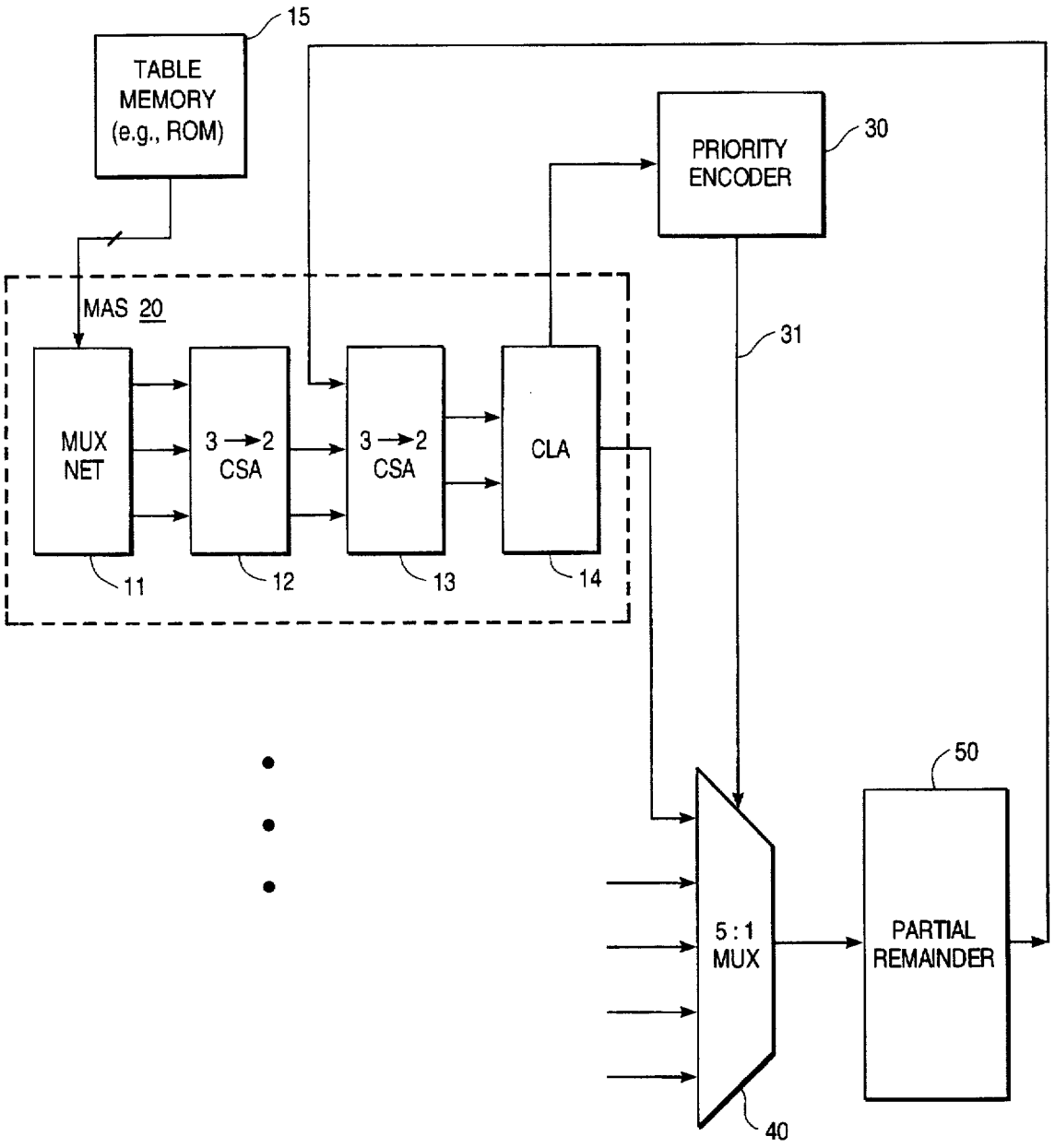

15
TABLE MEMORY (e.g., ROM)
PRIORITY ENCODER
30
MAS 20
31
MUX NET
3 → 2 CSA
3 → 2 CSA
CLA
11
12
13
14
50
5 : 1 MUX
PARTIAL REMAINDER
40

FIG_6

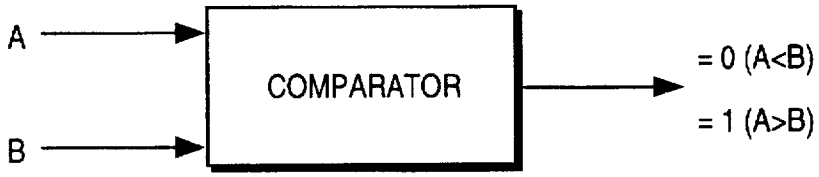


FIG. 7

DIVISION ALGORITHM FOR FLOATING POINT OR INTEGER NUMBERS

FIELD OF THE INVENTION

This invention relates generally to the field of computer arithmetic; specifically, to computer implemented methods and circuits for dividing floating point or integer numbers.

BACKGROUND OF THE INVENTION

Computer implemented algorithms for performing arithmetic operations have existed since the advent of computers. Most often, these algorithms represent a sort of binary version of the paper-and-pencil method taught in elementary school. Many of the algorithms differ according to the particular number system that is employed. For example, there are four common representations for N-bit numbers: signed magnitude, two's complement, one's complement, and bias. As one would expect, algorithms also differ based upon the type of operation performed (e.g., addition, subtraction, multiplication, division), the precision to be implemented, exception handling, and so on.

Division algorithms are generally classified into two types, restoring and non-restoring. Examples of both restoring and non-restoring types of division algorithms can be found in the book, "Computer Architecture—A Quantitative Approach", Second Edition, by Patterson and Hennesy, Appendix A, Morgan Kaufmann Publishers, Inc. (1996).

As practitioners in the field understand, restoring division algorithms get their name because when a subtraction step involving the divisor yields a negative result, the register containing the remainder is restored to its old value. Conversely, in non-restoring type division algorithms, the remainder does not have to be restored at any stage of the calculation.

Much effort in the field of computer arithmetic has been devoted to simplifying or minimizing the number of operations that must be performed. By way of example, in non-restoring SRT division an arithmetic logic unit (ALU) operation is normally performed at each step. (SRT is an acronym for Sweeney, Roberson, and Tocher, who originally proposed algorithms of this nature). SRT division is widely employed in computer systems such as those which include the popular Pentium® processor, manufactured by Intel Corporation.

Despite its popularity, SRT division is not without its complications. For instance, one drawback of SRT division is that quotient bits cannot be determined immediately as they can in ordinary non-restoring division. Another drawback is that the quotient is calculated in a redundant representation that typically requires relatively complex logic. Furthermore, in SRT division negative quotient values are needed which further add to the computational logic. Obviously, complications in the compute logic add delay to the calculating steps.

As an alternative to existing SRT and multiplicative algorithms, the present invention presents a computer that implements a novel division algorithm of the non-restoring type. The algorithm is well-suited for dividing floating point or integer numbers and enables implementing robust, simple and fast dividers. Moreover, the dividers are easy to verify and provide high speed performance compared to many existing computer designs. Additionally, the logic for calculating the next group of quotient bits is simplified and permits iteratively calculating N-bits of the quotient per cycle.

SUMMARY OF THE INVENTION

The present invention covers a computer-implemented method and apparatus for dividing numbers in either floating-point or integer representation. In one implementation, the algorithm begins by first choosing either a zero or a one as a quotient digit to the left of the decimal point based upon a comparison of the divisor and the dividend. Next, the divisor is subtracted from the dividend to generate a first intermediate result. The first intermediate result is then shifted in a register by N-bits to obtain a remainder value.

After obtaining the remainder, a memory containing a table of multipliers is referenced. The table is indexed by a portion of the remainder and a portion of the divisor. Referencing the table produces one or more multipliers.

The next step in the algorithm involves multiplying the divisor by each of the previously referenced multipliers. This generates corresponding second intermediate results. Each of the second intermediate results are then subtracted from the remainder to generate corresponding third intermediate results. The current quotient digit is selected as the largest multiplier which corresponds to the third intermediate result having the smallest positive value (as among all of the third intermediate results). These steps can be repeated to calculate additional quotient digits. For each iteration, the third intermediate subtraction result that corresponds to the selected multiplier is used as the partial remainder in the next iteration.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be understood more fully from the detailed description which follows and from the accompanying drawings, which however, should not be taken to limit the invention to the specific embodiments shown, but rather are for explanation and understanding only.

FIGS. 1A–1C illustrate various terms used in the field of computer arithmetic; particularly, in performing division.

FIG. 2 shows the relationship between the partial remainder upper bits and the upper divisor bits used to evaluate the next quotient digit in accordance with the present invention.

FIG. 3 is a diagram showing bit ranges of the partial remainder and divisor and the corresponding quotient digit sets for logic minimization in accordance with one embodiment of the present invention.

FIG. 4 a table which specifies formation of the divisor according to one embodiment of the present invention.

FIG. 5 is a simple numerical example of the steps involved in one possible implementation of the invented division algorithm.

FIG. 6 is a block diagram of a circuit for dividing two numbers according to the present invention.

FIG. 7 shows a comparison of two numbers to generate a leading digit according to one embodiment of the present invention.

DETAILED DESCRIPTION

A method and apparatus for performing floating point or integer division is presented. The method enables implementing robust, simple, fast dividers that is suitable for performing remainder instructions as well. The proposed algorithm is of the non-restoring type, meaning that the remainder does not have to be restored at any stage in the calculation. Unlike common SRT algorithms, the quotient is not calculated in a redundant representation. This simplifies

and speeds up the logic required for calculating the next group of quotient bits. The proposed algorithm is iterative, calculating N-bits of the results, per cycle, where N is an area/speed tradeoff.

Referring now to FIGS. 1A–1C, there are illustrated terms and numerical representations that will be used throughout the remainder of this specification. FIG. 1A shows a simple division example involving three digits: 24 is the dividend, which is divided by 8, the divisor. The result of this calculation is the number 3, i.e., the quotient digit.

FIG. 1B illustrates a non-integer representation that has gained widespread use in the field of computer arithmetic. This representation is known as floating point. In floating point representation, the computer word or number is broken into two parts: an exponent and a significand. As can be seen in FIG. 1B, the significand is the number 1.50, while the exponent is −3. The result of this calculation is 0.1875. The decimal part of a fractional number is commonly referred to as the mantissa.

The example of FIG. 1C illustrates how most floating point division algorithms are carried out. That is, the quotient is typically calculated by dividing the two significands, with the exponent portion being calculated by a simple subtraction. Thus, in the example of FIG. 1C the quotient value is given by (S_1/S_2), whereas the exponent portion is simply $2(e_1-e_2)$.

Although the present invention will be described in particular embodiments utilizing floating point representations, it should be understood hat the present invention is also applicable to integer division. Integer division may be performed in accordance with the present invention by first normalizing the integer values, and then proceeding with the steps described below. Practitioners in the art will further appreciate that the proposed division algorithm is well-suited for a variety of floating point representations, including the format specified by IEEE standard 754-1985 (also International Standard, IEC 559). This format is described in "A Proposed Radix-Hand Word-Length-Independent Standard for Floating-Point Arithmetic" IEEE Micro 4:486-100, Cody, et al. (1984).

The invented division algorithm is based on the following iterative formula.

$$Pr(i+1)=\text{RADIX}*Pr(i)-D*Q(i+1)$$

where Pr(i) is the partial remainder in the i'th iteration, D is the divisor and Q(i+1) is the quotient digit currently being calculated. An example of this algorithm is demonstrated below (see FIG. 5) in a version which calculates four quotient bits per clock (radix 16). It should be appreciated, however, that the algorithm is quite generalized and can be extended to more (or less) bits per cycle. The calculation is iterative and continues until the required precision has been achieved or the remainder reaches zero. The latency is thus the width of the mantissa divided by four. (An additional cycle is typically needed for the rounding action.)

According to the invention, at each clock cycle of the computer system the upper bits of the partial remainder are inspected along with the upper bits of the divisor to determine the possible values of the next quotient digit. The number of bits that are inspected depends upon the maximum number of possible values permitted. In the example provided below, four most significant bits (MSBs) are inspected for the partial remainder and two MSBs are inspected for the divisor. This yields up to five possible quotient digit values. Note that since the divisor is typically normalized it is always in the range of 1–2. This means that its most significant bit is always one, so that the actual bits inspected are the two bits immediately following the decimal point (i.e., to the right of the decimal point).

Those persons familiar with computer arithmetic will appreciate that tradeoffs can be made between the total number of bits inspected from the divisor and from the partial remainder, as well as the number of possible quotient digits out of the radix possible.

With reference now to FIG. 2, there is shown graphically the relationship between the partial remainder upper bits and the upper divisor bits. The horizontal scale is for a normalized divisor, as in a floating point divider where the divisor is always in the range of 1–2. Of course, for integer dividers, the divisor would be pre-normalized to fall within such a range. The vertical divisions in the graph of FIG. 2 correspond to the number of divisor MSBs chosen. In the example shown, two most significant bits are to be inspected to evaluate the next quotient. Thus, there are four distinct divisor regions corresponding to the bits 00, 01, 10 and 11. By restricting the number of bits that are inspected, the size of the table shown in FIG. 2 is restricted and the speed at which the calculation is performed is increased.

The vertical scale for the radix, for example, extends from 0 to 32 (as the partial remainder lies in the range of $0–2^1$). After being multiplied by the radix (16 in our example) as per the above equations, this covers the range 0–32. The divisions on the vertical scale (16 of them) correspond to the number of partial remainder MSBs (4 bits) chosen for the purpose of evaluating the next quotient. Hence, the sixteen vertical regions correspond to the binary numbers 0000–1111, as shown along the right hand side of FIG. 2.

The possible quotient digit results are shown in FIG. 2 as being from 0–15. These are the lines which mark the regions for each value of the quotient bits. For instance, the line which extends from the partial remainder value 1 to the partial remainder value 2, and across the divisor upper bit values 1–2, marks the area where the correct quotient value is zero. Similarly, the area between the lines marked 1 and 2 (along the vertical axis) is the one in which the divisor and remainder values produce a quotient digit of 1. The area above the line extending from a partial remainder value of 15 to 30, and across the full range of the divisor bits, produces a correct quotient of 15.

During each iteration of the invented division algorithm, a table containing information such as that shown in FIG. 2 is accessed to determine how many times the divisor "fits" into the present or current remainder. Consider, for example, the case in which the upper partial remainder bits are 0010 and the divisor upper bits are 01. A table look-up (e.g., to a ROM containing the values shown in FIG. 2) produces probable quotient digit results of 2, 3 or 4. This is shown by the shaded region in FIG. 2. In other words, because a decision has been made to pick the correct quotient value based upon reduced precision versions of the divisor and the partial remainder, the shaded region of FIG. 2 encompasses several values of the quotient that may provide the correct quotient digit value. In this respect, each of the rectangular regions of FIG. 2 can be thought of as representing a zone of uncertainty, since at this stage of the algorithm it is not known exactly which quotient will fit into the remainder without producing a negative next remainder.

The regions are designed so that, at most, up to N possible candidates are selected. The number of candidates is the function of the number of bits used in the look-up, and can be minimized by judicious selection of the regions. According to one embodiment of the invention, up to five values are determined in each clock. These values represent all of the possible (i.e., a superset) quotient digits for the current iteration.

Once all of the five possible quotient bit-values have been generated from the table look-up, all five multiples of the divisor by those values are calculated, and then subtracted from the current partial remainder in a parallel operation. The result yielding the smallest positive result is then chosen as the correct value for the current quotient digit. The corresponding subtraction result is the new partial remainder for the next iteration. To restate it differently, the algorithm picks the largest multiple (i.e., multiplier) from the table (e.g., FIG. **2**) that still provides a positive result following subtraction. Based on the number of bits that are inspected for both the partial remainder and the divisor, the table look-up generates all of the quotient bit values that—when multiplied by the divisor—are likely to fall within the quotient domain.

FIG. **3** illustrates quotient digit sets for various values of the partial remainder and divisor according to one embodiment of the present invention. As can be seen, five possible values for the next digit are always provided by the table represented by FIG. **3**—even when the actual range of possible values may be smaller. For instance, in some situations a zero ("0") multiplier is used, whereas in other cases, redundant values are input. This embodiment permits minimization in the control logic and minimization of the required multiplexer network. In other words, even though only one or two values need be provided (depending upon a particular partial remainder and divisor values), five multipliers are always generated (see FIG. **3**) so that various logic minimizations can be achieved. Note that in FIG. **3**, different zones are combined into larger areas. By way of example, whenever the partial remainder upper bits falls in the range 1100–1111, the values 0, 12, 13, 14, 15 are always produced, irrespective of the divisor MSBs.

To better understand how division is performed according to the present invention, consider the example presented in FIG. **5**. In this example it is desired to divide A (=1.5) by B (=1.25). Both the decimal and binary representations of these values are shown in FIG. **5**. The initial step involves comparing the values of A and B to choose either 0 or 1 as the leading digit (to the left of the decimal point). This selection can be made simply by comparing A to B to determine which is larger, as shown in FIG. **7**. In this case, A is greater than B, so 1 is chosen as the leading or first digit.

In the second step, the divisor is subtracted from the dividend to generate an unshifted remainder. In our example, this subtraction step is shown below.

$$\begin{array}{r} 1.1000 \\ \underline{1.0100} \\ 0.0100 \end{array}$$

Next, the remainder is shifted by 4-bit locations—basically multiplying the remainder by 16 in accordance with the radix of our computation. As can be seen below, this shifting step produces the remainder value 0100.000.

```
00000.010   (before multiplying by the radix)
00100.000   (after shifting left by 4)
{0010}      --> the bits used in the look-up.
```

After shifting, a memory containing the look-up table values is accessed to obtain the multipliers containing the correct quotient digit for the current iteration. Referring to the graph of FIG. **2**, the partial remainder and divisor, for example, would produce values 2, 3 and 4. One of these values represents the correct quotient digit for the current iteration.

After the multipliers are obtained from the look-up operation each value is multiplied by the divisor, with the result being subtracted from the remainder. This yields three corresponding intermediate results. In our example, two of the results are positive and one is negative. According to the invented division algorithm, the largest multiplier that yields the smallest positive result is selected as the quotient digit. Applying this selection rule to the example of FIG. **5** means that "3" is selected as the quotient digit for the current iteration. The next partial remainder is calculated using the selected quotient, and the same process is repeated for the next iteration.

Practitioners in the art will appreciate that the proposed division algorithm offers a number of advantages compared with conventional SRT algorithms. One benefit is that the redundant representation of the quotient is avoided. Quotient bits are calculated directly, therefore they do not need to be later calculated into a final non-redundant form.

Look-up tables for the quotient selection are also smaller, with a whole range of possible tradeoffs in the selection and number of bits used for the lookup. Because the algorithm is numerically simple, error analysis is likewise easier to perform.

Perhaps the greatest advantage over conventional SRT algorithms is that the operating frequency according to the present invention is superior when a large number of quotient bits (e.g., more than three) are to be calculated per cycle. This is primarily due to the smaller number of bits used in the next quotient look-up table and the larger parallelism inherent in the algorithm. Compared with other prior art algorithm, such as multiplicative algorithms, the present invention may be implemented via a relatively simple network of multiplexers and adders. Because the algorithm is numerically simple, it also does not require expert numerical analysis or the use of a relatively large multiplier array.

One possibility for generating the quotient times divisor multiples is shown in the table of FIG. **4**. All of the multiples of the divisor are produced in FIG. **4** by a summation of up to three shifted versions of the divisor. Thus, each of the multiples 0–15 can be produced by summing first, second and third summands. By way of example, the multiple 13 is produced by summing the divisor with the divisor shifted left by two places, and the divisor shifted left by three places. Mathematically this may be written as:

$$D\times 13 = D + D\times 4 + D\times 8$$

The table of FIG. **4** specifies how each multiple of the divisor may be formed in accordance with one embodiment of the invention. The numbers in the table denote multiples by that particular number. For example, the multiple "8" means **8***D (i.e., eight times the divisor). Practitioners in the art will readily appreciate that multiples by powers of two are relatively easy to generate as they are produced by simple shifts. Multiples by 14 and 15 are performed by subtraction from the multiple of 16. Subtraction is performed by addition of the inverted multiple (i.e., 1# or 2#) and the binary number one (i.e., '1).

FIG. **6** is a circuit schematic block diagram showing one possible implementation of the division algorithm of the present invention. The illustrated embodiment selects five multiples in accordance with the look-up information shown in FIG. **3**. These five multiples are produced by five multiply-and-subtract (MAS) units **20**. In the embodiment of FIG. **6**, 2 carry-save adders (CSAs) **12** and **13** are employed along with a carry-look-ahead adder (CLA) **14**. (Carry-save adders are well known circuits that comprise a collection of

N independent full adders). Each addition operation results in a pair of bits: a sum, and a carry bit.

To form the various multiples shown in the table of FIG. **4**, up to three numbers need to be added (radix 16). Hence, a three input adder **12** is utilized with its three inputs being coupled to a multiplexer network **11** that produces the first, second, and third summands. Multiplexer network **11** coupled to memory **15** provides the summand values listed in the table of FIG. **4** as inputs to carry-save adder **12**. It should be apparent that the multiplexer for the first and third summand may be implemented using a 4:1 multiplexer stage, while a 5:1 multiplexer will suffice to produce the second summand.

The two stages of the 3:2 CSAs **12** and **13** are connected in series with CLA **14**. CSA **12** produces the multiple of the divisor in a sum and carry form that is connected to the inputs of the CSA **13**. The third input to CSA **13** is the current partial remainder of the iteration.

In FIG. **6**, the partial remainder is shown being stored in a register **50**. Thus, CSA **13** subtracts the multiple of the divisor from the partial remainder and again produces a sum and carry pair as outputs coupled to the inputs of CLA **14**. The CLA, in turn, produces the next partial remainder, which is coupled to one input of a 5:1 multiplexer **40**. Each MAS **20** produces a potential next partial remainder that corresponds to the multiple generated from the look-up. Each of the CLAs also produce a carry out signal that is input to the priority encoder **30**.

Priority encoder **30** controls which partial remainder to select next, i.e., which result goes into the partial remainder register **50** for the next iteration.

This is shown in FIG. **6** by control line **31** coupled to the select input of multiplexer **40**. Priority encoder **30** also selects which quotient digit is selected as the correct digit for the current iteration. To make the quotient digit selection, priority encoder **30** controls another multiplexer (not shown in FIG. **6**) that selects the correct quotient digit to be loaded into the appropriate location in the quotient result register. The quotient result register is shifted each iteration by the number of quotient bits computed, so that the currently computed bits are always inserted at the least significant bit locations. As explained previously, the correct quotient for the current iteration is the largest one that produces a multiple that still provides a positive result following subtraction from the remainder. Subtracting the quotient digit from the remainder produces the next partial remainder for the next iteration.

Practitioners in the art will appreciate that the present invention is flexible enough to compute a variety of different numbers of bits each clock cycle. For instance, if one wanted to compute five bits of the result per cycle—as may be the case in computing radix 32 computations—a circuit would be needed that could produce multiples ranging from 0 to 31. This could be achieved, for example, by summing four different numbers together.

Another possibility is to reduce the number of multipliers by choosing finer granularity in the table. In this respect, if three bits were used as the upper bits of the divisor, then the size of the columns (see FIG. **2**) would be reduced in half. This, in turn, would reduce the number of multiples that need to be generated.

The division algorithm of the present invention therefore allows a tradeoff: fewer bits may be used for both the partial remainder and divisor with faster look-up table access, but at the expense of more multiples. On the other hand, more bits may be used to reduce the number of multiples produced for each iteration, but at the expense of slower look-up circuitry. The benefits of the proposed division algorithm, however, provide advantages in both speed and circuit size over previous division methods.

I claim:

1. A computer-implemented method of dividing numbers according to a radix comprising the steps of:

(a) subtracting a divisor from a dividend to generate a first intermediate result in a storage location having a plurality of bits;

(b) shifting the first intermediate result by N-bits, where N is an integer and 2^N is equal to the radix, to obtain a remainder;

(c) referencing a memory unit containing a table of multipliers that is indexed by a portion of the remainder and a portion of the divisor, the table providing one or more multipliers;

(d) multiplying the divisor by each of the one or more multipliers to generate one or more second intermediate results;

(e) subtracting from the remainder each of the one or more second intermediate results to generate one or more corresponding third intermediate results;

(f) selecting as a quotient digit a largest multiplier from the one or more multipliers which corresponds to a third intermediate result having a smallest positive value as among the one or more third intermediate results.

2. The method of claim **1** further comprising the initial step of:

choosing either 0 or 1 as a quotient value to the left of a decimal point based upon a comparison of the divisor and the dividend.

3. The method of claim **2** wherein the choosing step comprises the step of:

choosing 1 as the quotient value to the left of the decimal point when the dividend is greater than or equal to the divisor.

4. The method of claim **2** wherein the choosing step comprises the step of:

choosing 0 as the quotient value to the left of the decimal point when the dividend is less than the divisor.

5. The method of claim **1** wherein N is equal to 4.

6. The method of claim **5** wherein step (c) provides five multipliers from the table of multipliers.

7. The method of claim **1** wherein step (c) provides up to five multipliers from the table of multipliers.

8. The method of claim **1** wherein the dividend and divisor are integer numbers and further comprising the step of:

normalizing the dividend and the divisor.

9. The method of claim **1** wherein the portion of the remainder comprises an upper N-bit portion of the remainder.

10. The method of claim **1** wherein the portion of the divisor comprises an upper (N/2)-bit portion of the divisor.

11. The method of claim **10** wherein N=4.

12. The method of claim **1** further comprising the step of:

calculating a next quotient digit by repeating steps (b)–(f) utilizing the third intermediate result from step (f) as the first intermediate result.

13. The method of claim **1** wherein the memory unit comprises a read-only memory (ROM).

14. A computer apparatus for dividing numbers according to a radix comprising:

a storage location having a plurality of bit positions;

means for subtracting a divisor from a dividend to generate a first intermediate result in the storage location;

means for shifting the first intermediate result in the storage location by N-bits, where N is an integer and 2^N is equal to the radix, to obtain a remainder;

a read-only memory (ROM) containing a table of multipliers that is indexed by a portion of the remainder and a portion of the divisor, the table providing one or more multipliers;

means for multiplying the divisor by each of the one or more multipliers to generate one or more second intermediate results;

means for subtracting from the remainder each of the one or more second intermediate results to generate one or more corresponding third intermediate results;

means for selecting as a quotient digit a largest multiplier from the one or more multipliers which corresponds to a third intermediate result having a smallest positive value as among the one or more third intermediate results.

15. The computer apparatus of claim **14** further comprising:

means for choosing either 0 or 1 as a quotient value to the left of a decimal point based upon a comparison of a divisor and dividend.

16. The computer apparatus of claim **15** wherein 1 is chosen as the quotient value to the left of the decimal point when the dividend is greater than or equal to the divisor.

17. The computer apparatus of claim **15** wherein 0 is chosen as the quotient value to the left of the decimal point when the dividend is less than the divisor.

18. The computer apparatus of claim **14** wherein N=4.

19. The computer apparatus of claim **18** wherein the table of multipliers provides five multipliers.

20. The computer apparatus of claim **14** the table of multipliers provides up to five multipliers.

21. The computer apparatus of claim **14** wherein the dividend and divisor are integer numbers.

22. The computer apparatus of claim **14** wherein the portion of the remainder comprises an upper N-bit portion of the remainder.

23. The computer apparatus of claim **14** wherein the portion of the divisor comprises an upper (N/2)-bit portion of the divisor.

24. The computer apparatus of claim **23** wherein N=4.

25. The computer apparatus of claim **14** further comprising:

means for computing a next quotient digit utilizing the third intermediate result generated by the selecting means as the first intermediate result coupled to the shifting means.

26. A computer apparatus for performing division comprising:

a register for storing a remainder which is a N-bit shifted result of subtracting a divisor from a dividend;

a memory containing a table of multipliers that is indexed by a portion of the remainder and a portion of the divisor the memory providing K multipliers, where K is an integer greater than 1;

K arithmetic units, each of which is to the memory to receive a corresponding one of the K multipliers and the remainder stored in the register; during an iteration cycle, the arithmetic units producing K next remainders each of which is computed by

(i) multiplying the divisor by the corresponding one of the K multipliers to generate a first result;

(ii) subtracting the first result from the remainder; and

control logic, coupled to the K arithmetic units, that selects a current quotient digit as a largest one of the K multipliers which corresponds to a next remainder having a smallest positive value as among the K next remainders.

27. The computer apparatus of claim **26** wherein the control logic further selects the next remainder having a smallest positive value for storage in the register for a next iteration cycle.

28. The computer apparatus of claim **26** wherein K=5.

29. The computer apparatus of claim **28** wherein N=4.

30. The computer apparatus of claim **26** wherein each of the arithmetic units comprises a multiply-and-subtract unit.

31. The computer apparatus of claim **30** wherein each of the multiply-and-subtract units comprises:

a first carry-save adder (CSA) which receives the corresponding one of the K multipliers as a set of summands, the first CSA computing a multiple of the divisor therefrom; and

a second CSA coupled to receive the multiple of the divisor and the remainder, the second CSA subtracting the multiple of the divisor from the remainder.

32. The computer apparatus of claim **26** wherein the control logic comprises a priority encoder coupled to each of the K arithmetic units.

33. The computer apparatus of claim **31** wherein N=4 and K=5.

34. The computer apparatus of claim **26** wherein the memory comprises a read-only memory (ROM) device.

35. The computer apparatus of claim **26** further comprising:

a circuit that generates a quotient digit to the left of a decimal point based on a comparison of the dividend and the divisor.

36. The computer apparatus of claim **26** wherein the portion of the remainder comprises an upper N-bit portion of the remainder.

37. The computer apparatus of claim **26** wherein the portion of the divisor comprises an upper (N/2)-bit portion of the divisor.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,784,307

DATED : July 21, 1998

INVENTOR(S) : Gad S. Sheaffer

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In column 3 at line 29 delete "hat" insert --that--

Signed and Sealed this

Second Day of February, 1999

Attest:

Attesting Officer

Acting Commissioner of Patents and Trademarks