Advance Access publication on November 23, 2007
doi:10.1093/comjnl/bxm082

Efficient Algorithms for Integer Division by Constants Using Multiplication

D. Cavagnino* and A. E. Werbrouck

Dipartimento di Informatica, Università degli Studi di Torino, C.so Svizzera 185, 10149 Torino, Italy
**Corresponding author: davide@di.unito.it*

We present a complete analysis of the integer division of a single unsigned dividend word by a single unsigned divisor word based on double-word multiplication of the dividend by an inverse of the divisor. The well-known advantage of this method yields run-time efficiency, if the inverse of the divisor can be calculated at compile time, since multiplication is much faster than division in arithmetic units. Our analysis leads to the discovery of a limit to the straightforward application of this method in the form of a critical dividend, which fortunately associates with a minority of the possible divisors (20%) and defines only a small upper part of the available dividend space. We present two algorithms for ascertaining whether a critical dividend exists and, if so, its value along with a circumvention of this limit. For completeness, we include an algorithm for integer division of a unsigned double-word dividend by an unsigned single-word divisor in which the quotient is not limited to a single word and the remainder is an intrinsic part of the result.

Keywords: integer division; multiplicative inverse; division by multiplication; efficiency; integer constants

Received 19 January 2007; revised 23 August 2007

1. INTRODUCTION

It is quite normal that a general purpose computer architecture includes in its instruction set an integer multiplication of two single words to produce a double-word product. Such a product may be available as the destination of two separate instructions, one for the low half and the other for the high half, as in the Alpha [1], Itanium [2] and PowerPC [3] architectures, or can be produced by one instruction with the product in a HiLo pair as in the MIPS R10000 [4] architecture. Since division can be considered an inverse of multiplication, it naturally follows that the most general type of division is a double-word dividend divided by a single-word divisor to produce a single-word quotient and a single-word remainder. Such an operation is consistent with the classic scheme of an arithmetic logic unit (ALU) associated with a multiplier–quotient extension, known in the literature [5] as the MQ register.

Modern instruction sets do not usually include direct access to the ALU since the RISC approach requires all operands to be in an uniform set of general registers, which being independent do not foresee being coupled in pairs, the HiLo pair in MIPS R10000 being an exception. As a result, the dividend is frequently limited to a single word. With the extension to 64-bit words, this choice is adequate for many applications.

While multiplication can be rendered highly parallel, the same is not true for division. If the computer architect is satisfied with integer division limited to single-word dividends, then the multiplication of this single-word dividend by a proper multiplicative inverse of the divisor can produce quickly the quotient as the upper half of the ensuing product. The work involved in determining the multiplicative inverse is greater than that required in a direct integer division. Thus, this approach is attractive only if the divisor is known at compile time when the additional effort involved in determining a multiplicative inverse need to be expended by the compiler only once for each constant divisor. Two of the more recent architectures, Alpha [1] and Itanium [2], do not even have native integer division instructions. The Itanium even affects the integer multiplication only in the floating point unit.

We present a detailed analysis of a method to find the quotient in an integer division of an unsigned single-word dividend by an unsigned single-word divisor in which the principal operation is the multiplication of the dividend by the multiplicative inverse of the constant divisor known at compile time. For maximum efficiency, we try to limit the run-time operations to multiply long and shift right. In a few cases which we isolate, it is necessary to execute one (even divisor) or two (odd divisor) extra instructions all of which are non-special and no overflow is involved. Our derivation is quite different from the most frequently cited algorithm for this operation published in 1994 [6]; in which the cost of the unsigned division is one multiply, two add/subtracts and two shifts for each case; our algorithm requires in most cases a multiplication and one shift only (moreover, our multiplicative inverse fits in a single-word for all cases). The preliminary version of our single-word division was deposited in [7]. Our method produces only quotients. To obtain remainders, a multiplication and a subtraction are necessary. To be general, we present also a division of an unsigned double-word dividend by an unsigned single-word divisor for which the quotient is not limited to a single word as in [6], and the remainder is an intrinsic part of the result. Different approaches oriented towards division by small divisors are given in [8, 9]. The approach in [9] uses an optimization of [6] and as such remains different from our approach in that our algorithms require in execution only standard instructions whereas the method in [9] leads to some multiplicative inverses greater than the word size. An elegant algorithm for computing the division by a constant using multiplication is attributed to T. Mathisen in [10, Section 16.9]. This algorithm is very similar to ours. We show how our method encompasses cases in which our algorithm is still more efficient than the one in [10, Section 16.9] because in most cases we save a sum as we show later in detail.

An interesting combinatoric division algorithm for fixed integer divisors, which exploits the periodicity in the inverse of odd integers, is presented in [11]. However, it still requires a double-word product besides shift and add operations.

Since our analysis is valid for all common register widths W, we define symbolic variables for the derivations and then present numerical results for register widths of 32 and 64 bits. Some significant intermediate results are made explicit as powers of 2 parameterized by W.

2. ALGORITHMS AND ANALYSIS

Our division by multiplication (DBM) method computes the quotient Q in an integer division of an unsigned dividend N by an unsigned divisor D, i.e.

$$Q = \left\lfloor \frac{N}{D} \right\rfloor,$$

where the symbol $\lfloor\ \rfloor$ indicates the floor function that computes the largest integer not greater than its argument and D is a constant value that is *known at compile time*.

In our analysis, we exclude the cases when D is a power of 2 (i.e. $D = 2^n$), because in that case the division may be performed by simply right shifting the dividend by n positions.

To facilitate the comprehension of our analysis, we define the following useful variables that are employed throughout the paper and which have reasonable mnemonic values:

W: number of bits defining the word *Width* of the processor for which the code that performs the division is generated; F: 2^W, the number of values in a *Full* word, L: *Length* in bits of the significant part of the divisor, $2 \le L \le W$, S: 2^{W-L}, a *Scale* factor to shift the most significant bit (MSB) of the divisor to the leftmost position in a word, C: 2^{L-1}, the final divisor to *Clip* the upper word of the product, E: $2^{W-1} = CS$, an *Elementary* factor in the analysis, $R = N\text{-}QD$: *Remainder*.

$$I = \left\lfloor \frac{FE}{SD} \right\rfloor = \left\lfloor \frac{FCS}{SD} \right\rfloor = \left\lfloor \frac{FC}{D} \right\rfloor = \left\lfloor \frac{2^{W+L-1}}{D} \right\rfloor \qquad (1)$$

where I is the floor of the first approximation *Inverse*.

$J = I + 1$ is the multiplicative inverse value that is used to compute the quotient.

Note that $0 \le N \le F - 1 = 2^W - 1$ given that N is contained in a processor word, and that our divisor satisfies $2^{L-1} + 1 \le D \le 2^L - 1$, from the definition of L.

The value of I is limited by the largest and smallest possible values for SD. The largest value of SD is $2^{W-L}\ (2^L - 1) = 2^W - 2^{W-L}$, which becomes $2^W - 1$ as largest value when $L = W$. The smallest value of SD is $2^{W-L}\ (2^{L-1} + 1) = 2^{W-1} + 2^{W-L}$ which becomes $2^{W-1} + 1$ as smallest value when $L = W$.

The integer part of $I_{\min}$ is $2^{W-1} = E$, from

$$I_{\min} = \frac{FE}{2^W - 1} = \frac{2^W 2^{W-1}}{2^W - 1} = 2^{W-1}\left(\frac{1}{1 - 2^{-W}}\right)$$
$$= 2^{W-1}(1 + 2^{-W} + 2^{-2W} + \cdots) = 2^{W-1} + 2^{-1} + 2^{-W-1} + \cdots.$$

The integer part of $I_{\max}$ is $2^W - 2 = F\text{-}2$, from

$$I_{\max} = \frac{FE}{2^{W-1} + 1} = \frac{2^W 2^{W-1}}{2^{W-1} + 1} = 2^W\left(\frac{1}{1 + 2^{-(W-1)}}\right)$$
$$= 2^W[1 - 2^{-(W-1)} + 2^{-2(W-1)} - \cdots] = 2^W - 2^1 + 2^{-W+2} + \cdots.$$

The limits on I are $2^{W-1} \le I \le 2^W - 2$, so I is contained in a word of the processor, and its leftmost bit is 1. The same may be stated for $J = I + 1$, because $2^{W-1} + 1 \le J \le 2^W - 1$.

From the division in (1), we see that

$$DI < FC < D(I+1) = DJ \tag{2}$$

since $FC = 2^{W+L-1}$ is a power of 2 and D is not.

From the left-hand side of (2), we have $DI \leq FC - 1$, and from the right-hand side of (2), $FC + 1 \leq DI + D$. So, the limits on DI are

$$FC - D + 1 \leq DI \leq FC - 1. \tag{3}$$

The first approximation of the quotient Q is given by the function DBM_a(N, J) that has as arguments the dividend N (known at run-time) and the value J computed at the compile time:

```
DBM_a($N$, $J$)
    double word $H$;
    $H = N*J$;
    $Q' = H / (F*C)$;   // $FC = 2^{W+L-1}$
    return $Q'$;
End.
```

The first step is a multiplication of two machine words, giving a double-word result. The second step is a division by 2^{W+L-1} which means discarding the right W bits (i.e. discard the right word), and then shifting right the remaining word by $L - 1$ bits (because $C = 2^{L-1}$). The result is a very fast computation of the quotient, without performing any 'real' division. That is made possible because, in certain sense, the compiler made the division work by computing J. In the following section, we examine the correctness of the result produced by DBM_a, that is $Q = Q'$, and give some adjustments due to particular cases to apply during run-time computation.

2.1. Discussion and proofs

The aim of this demonstration is to determine when the value computed by DBM_a, that is $NJ/(FC)$, is exactly the quotient Q. We obtain this result by substituting upper and lower limits on $NJ/(FC) = NJ / 2^{W+L-1}$.

The proper remainder of the division is

$$0 \leq R \leq D - 1 \tag{4}$$

thus, multiplying by I

$$0 \leq RI \leq (D-1)I. \tag{5}$$

However, from (3), $DI \leq FC - 1$, and thus

$$0 \leq RI \leq FC - 1 - I. \tag{6}$$

From $N = QD + R$, we have

$$\begin{aligned} NJ &= N(I+1) = NI + N = (QD+R)I + N \\ &= QDI + RI + N. \end{aligned} \tag{7}$$

We obtain an upper and a lower limit on NJ substituting these limits on DI and R in (7). When $R = 0$, we have

$$NJ = QDI + N = QDI + QD. \tag{8}$$

But given that from (3) $FC - D + 1 \leq DI$, then

$$Q(FC - D + 1) + QD = Q(FC+1) \leq NJ. \tag{9}$$

An upper limit for NJ may be obtained from the limits for RI and DI, respectively, in (6) and (3).

$$\begin{aligned} NJ &= QDI + RI + N \leq Q(FC-1) + [FC - (I+1)] + N \\ &= QFC - Q + FC - J + N = (Q+1)FC + (N - J - Q). \end{aligned} \tag{10}$$

Applying together the inequalities in (9) and (10), and dividing all members by $FC = 2^{W+L-1}$, we obtain a limit for the result of DBM_a:

$$Q + \frac{Q}{FC} \leq \frac{NJ}{FC} \leq (Q+1) + \frac{N - J - Q}{FC}. \tag{11}$$

From $0 \leq N \leq F - 1$, we have $(N - J - Q)/(FC) < 1$.

Given that the result is contained in a word, the lower limit correctly bounds Q' to Q because the division by $F = 2^W$ will make $Q/(FC)$ vanish from the upper word.

The upper limit requires a bit more analysis; in case the factor $(N - J - Q)/(FC)$ is non-negative, we cannot be certain that Q' is upper limited by Q (to ensure the correctness of DBM_a) because the upper limit becomes $Q + 1$.

If $N \leq E$ as is true for all signed divisions where $|N| \leq 2^{W-1}$, and given that $J > 2^{W-1}$, we have $(N - J - Q)/(FC) < 0$. This makes the upper limit less than $Q + 1$, and given that the result is an integer, the only possible (correct) value of DBM_a is Q.

We also note that the result produced by DBM_a is correct in the case $0 \leq R \leq D - 2$. In fact, the upper limit for NJ when $0 \leq R \leq D - 2$ is obtained by the following inequalities:

$$\begin{aligned} &0 \leq R \leq D-2, \quad \text{then} \quad 0 \leq RI \leq (D-2)I \quad \text{or} \\ &\quad 0 \leq RI \leq DI - 2I, \quad \text{and} \quad 0 \leq RI \leq FC - 1 - 2I, \\ &NJ = QDI + RI + N \leq Q(FC-1) + FC - 1 - 2I + N, \text{ thus} \\ &\quad NJ \leq (Q+1)FC + (N - 2I - Q - 1). \end{aligned}$$

In that case, we may write:

$$\frac{NJ}{FC} \le (Q+1) + \frac{N-2I-Q-1}{FC}. \tag{12}$$

Now, the quantity $N - 2I - Q - 1$ is always negative. The reason is that $0 \le N \le 2^W - 1$, and from $2^{W-1} \le I$, we infer that $2^W \le 2I$, thus $N - 2I$ is negative, as is $N - 2I - Q - 1$. The consequence is again that the upper limit is less than $Q + 1$, and given that the result is an integer, the only possible (correct) value of DBM_a is Q.

When the remainder of the division N/D is $D - 1$, there may be cases (not foreseeable at compile time) in which the result produced by DBM_a is wrong, due to the fact that the factor $(N - J - Q)/(FC)$ is non negative (i.e. ≥ 0), leaving an upper limit for DBM_a that is greater than or equal to $Q + 1$. These cases are determined by the value of N. That is, there is a minimum value of N, let us call it N_{cr} (N critical), for which

$$\begin{gathered}\frac{N_{\mathrm{cr}}J}{FC} \ge Q_{\mathrm{cr}} = Q+1,\\ N_{\mathrm{cr}} = QD + (D-1) = Q_{\mathrm{cr}}D - 1,\end{gathered} \tag{13}$$

where Q_{cr} is the corresponding wrong quotient.

Note that, from the considerations in the previous paragraph, N_{cr} has true remainder $D - 1$ if divided by D. We used Q_{cr} in the expression of N_{cr} because we are stating that DBM_a(N_{cr}) results in a wrong quotient $Q_{\mathrm{cr}} = Q + 1$ instead of Q only when the remainder is $D - 1$. Considering both expressions in (13), we may write

$$\frac{(Q_{\mathrm{cr}}D-1)J}{FC} \ge Q_{\mathrm{cr}} \tag{14}$$

leading to

$$Q_{\mathrm{cr}} \ge \frac{J}{DJ - FC}. \tag{15}$$

N_{cr} is the minimum dividend for which DBM_a produces a wrong result. For all the $N = (Q_{\mathrm{cr}} + k)D - 1 = N_{\mathrm{cr}} + kD$, $k \ge 0$, DBM_a gives the wrong result $Q_{\mathrm{cr}} + k$. In fact, from (2) and (13), we have

$$\frac{(N_{\mathrm{cr}}+kD)J}{FC} = \frac{N_{\mathrm{cr}}J}{FC} + \frac{kDJ}{FC} > Q_{\mathrm{cr}} + k, \quad k > 0.$$

2.2. Determining N_{cr} by division

From the previous discussion, we see that N_{cr} may be computed as

$$\begin{gathered}Q_{\mathrm{cr}} = \left\lceil \frac{J}{DJ-FC} \right\rceil,\\ N_{\mathrm{cr}} = Q_{\mathrm{cr}}D - 1.\end{gathered} \tag{16}$$

From (2), we see that the denominator of Q_{cr} is positive. We also see that N_{cr} is a function of D, and thus, when D is known at compile time, it is possible to determine N_{cr} for a given value of F (normally for $W = 32$ or 64). We will present an alternative algorithm to compute N_{cr} in the following section. As demonstrated in the appendix, if the fractional part of FC/D is greater than or equal to 0.5, a critical dividend (less than 2^W) cannot exist. If less than 0.5, a critical dividend can exist in our framework. An alternate solution presented in [10, Section 16.9] is analysed in the appendix.

For $N \ge N_{\mathrm{cr}}$, our general solution requires that N be decremented, even though it is intrinsically necessary *only* if $N = (Q_{\mathrm{cr}} + k)D - 1$, $k \ge 0$. Since we know the quotient and the remainder *only* after determining them, we always decrement $N \ge N_{\mathrm{cr}}$ and proceed to show that in all cases the result is correct as follows.

Suppose $N_{\mathrm{cr}} = Q_{\mathrm{cr}}D - 1$, and $Q_{\mathrm{cr}} = Q + 1$ as in (13). Note that DBM_a(N_{cr}) $= Q_{\mathrm{cr}}$, a wrong result, because $N_{\mathrm{cr}}/D = Q$ with remainder $D - 1$. Thus, Q_{cr} is the smallest quotient value that is not the correct quotient only when the true remainder is $D - 1$ and $N \ge N_{\mathrm{cr}}$. If $N \ge N_{\mathrm{cr}}$, we avoid this situation by decrementing N by 1. We give a proof of the correctness of this method for all remainders in the following.

(i) In case N/D has null remainder ($R = 0$) and $N > N_{\mathrm{cr}}$ we may write

$$N = (Q_{\mathrm{cr}} + k)D, \quad k \ge 0$$

while

$$\begin{aligned}N - 1 = (Q_{\mathrm{cr}}+k)D - 1 = (Q_{\mathrm{cr}}+k-1)D + (D-1),\\ k \ge 0.\end{aligned}$$

DBM_a($N - 1$) produces the correct quotient for the original dividend N (given that the dividend $N - 1$ has remainder $D - 1$ when divided by D).

(ii) Let us then consider the case N/D having a finite remainder ($R \ne 0$) and $N > N_{\mathrm{cr}}$. This implies

$$(Q_{\mathrm{cr}}+k)D + 1 \le N \le (Q_{\mathrm{cr}}+k)D + (D-1), \quad k \ge 0.$$

Let us decrement N by 1:

$$(Q_{\mathrm{cr}}+k)D \le N-1 \le (Q_{\mathrm{cr}}+k)D+(D-2), \quad k \ge 0.$$

For remainders $0 \le R \le D-2$, we have already shown in (12) that DBM_a returns the correct result, in this case $Q_{\mathrm{cr}}+k$. This is the correct quotient for N/D.

(iii) It remains to examine the case $N = N_{\mathrm{cr}}$ (not comprised in the two previous cases).

$$N = N_{\mathrm{cr}} = Q_{\mathrm{cr}}D - 1$$

implying that

$$\left\lfloor \frac{N}{D} \right\rfloor = Q_{\mathrm{cr}} - 1$$

DBM_a$(N-1)$ produces a correct quotient because its argument $N-1 = Q_{\mathrm{cr}}D-2$ is less than N_{cr}. Thus, DBM_a$(N-1) = Q_{\mathrm{cr}}-1$, and this is also the correct quotient for N/D.

In case the value of N_{cr} is less than 2^W, the function DBM may be written as:

```
DBM($N$, $N_{cr}$, $J$)
  If ($N >= Ncr$) Then $N = N - 1$; Endif;
  return DBM_a($N$, $J$);
End.
```

where N_{cr} is computed as in (16).

To illustrate our solution, we present in Fig. 1 a simplified example for word size of $W = 8$ bits, and $D = 11$ which implies that:

$$F = 2^8, \quad C = 2^{4-1} = 2^3, \quad FC/D = 186.1818, \quad I = 186, \quad \times J = 187, \quad Q_{\mathrm{cr}} = 21, \quad N_{\mathrm{cr}} = 230.$$

Note that for $N < N_{\mathrm{cr}}$, the floor of $NJ/(FC)$ corresponds to the correct quotient Q (symbols are superposed), while for $N \ge N_{\mathrm{cr}}$, the floor of $NJ/(FC)$ is wrong when the remainder is $D-1$, that is for $N = 230$, 241 and 252. By examining the symbols representing $(N-1)J/(FC)$, one sees that the floor of $(N-1)J/(FC)$, yields the correct quotient for $N \ge N_{\mathrm{cr}}$.

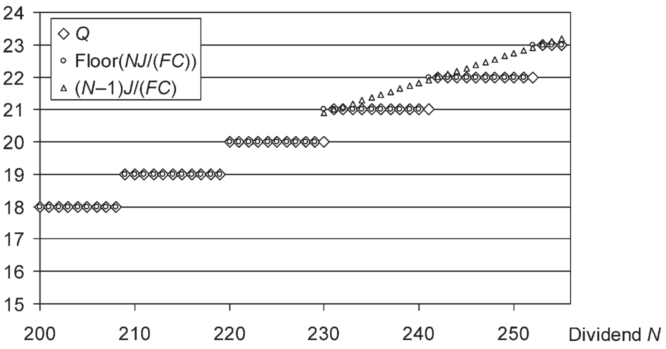


FIGURE 1. A simplified example of the effect of N_{cr} and its correction.

2.3. Further optimization

It is possible to introduce an optimization to the definition of DBM by analysing D. When D is even, then $N = QD + R$ will be:

(i) even, if R is even, namely $0 \le R \le D-2$, a remainder that ensures that DBM_a will not produce any errors;
(ii) odd, if R is odd, namely $1 \le R \le D-1$, and when $R = D-1$, the DBM_a may produce a wrong quotient. Setting the least significant bit, LSB(N) = 0 will solve the problem, forcing the remainder in the range $0 \le R \le D-2$, without influencing the quotient value.

The compiler may then discriminate the case when *D is even* and $N_{\mathrm{cr}} < 2^W$, and use the following definition of DBM:

```
DBM ($N$, $J$)
  Set LSB($N$) = 0;
  return DBM_a($N$, $J$);
End.
```

2.4. Code generation

Now we may write the various cases the compiler may face. We do not consider the special cases $D = 0$ and $|D| = 1$.

If $D = 2^x$, then generate the following code:

```
Right shift $N$ by x positions
```

Otherwise, in case $N_{\mathrm{cr}} \ge 2^W$ generate the following code:

```
DBM ($N$, $J$)
  return DBM_a($N$, $J$);
End;
```

Otherwise, in case D is even and the calculation of N_{cr} yields $N_{\mathrm{cr}} < 2^W$, generate the following code:

```
DBM ($N$, $J$)
  $N$=$N$ AND ($2^W - 2$); //Sets the LSB of $N$ to 0
  return DBM_a($N$, $J$);
End;
```

At this point, we are left with the case D is odd and $N_{\mathrm{cr}} < 2^W$, then generate the following code:

```
DBM ($N$, $Ncr$, $J$)
  If ($N >= Ncr$) Then $N=N-1$; Endif;
  return DBM_a($N$, $J$);
End;
```

2.5. Determining N_{cr} by multiplication

We have shown an analytical method (16) for determining N_{cr}. We here present an algorithm that, given D, computes N_{cr}

without involving division. This algorithm is based on the following considerations:

(i) $N_{\text{cr}} = Q_{\text{cr}}\,D - 1$ (i.e. N_{cr}/D gives remainder $D - 1$);
(ii) Q_{cr} has a maximum of $W - L + 1$ significant bits (given that D has L significant bits, and N is at most W bits wide, from $QD \leq N$, we have that Q can only be, at most, $W - L + 1$ bits wide);
(iii) for every $N \geq N_{\text{cr}}$, $N = kD - 1$, $k > 0$, DBM_a(N) responds with a wrong quotient.

The idea is to inspect the bits of the quotient starting from the MSB. A sketch of the algorithm is presented below.

```
Compute $J$ as previously specified;
$L$=ceil($\log_2(D)$); // $L$ is the number of significant bits of $D$
$Q$=0;
For i=$W - L$ to 0 step $-1$
   $Q$=$Q$+$2^i$; // Set the i-th bit.
   $N$=$QD - 1$; // The same as $N$=($Q - 1$)$D$+($D - 1$)
   If $N > 2^W - 1$ then $Q$=$Q - 2^i$; Next for; // Reset the i-th
         // bit and next for
   $Q_r$=DBM_a($N$, $J$); // Correct if Q − 1 (see value of $N$)
   If $Q_r$ <> ($Q - 1$) then $Q$=$Q - 2^i$; // Reset the i-th bit
End for;
$N_{cr}$=($Q$+1)$D - 1$;
If $N_{cr} > 2^W - 1$ then No $N_{cr}$;
```

The ith bit of Q is set to 1, then it is computed the value of N that, divided by D, should give a quotient $Q - 1$ and a remainder $D - 1$. Then, if N is not out of the $2^W - 1$ range, the function DBM_a is computed, and its result Q_{r} is compared with the value $Q - 1$: if they are equal, DBM_a computes correctly the quotient for N, meaning that N_{cr} may still be greater than N. Therefore, Q retains its current value, and will be possibly increased in further cycles. If DBM_a computes a wrong quotient (not equal to $Q - 1$) then N_{cr} is possibly smaller than (or equal to) N. Thus, smaller values will be inspected setting the ith bit of Q to 0.

After the loop, the value Q is the largest quotient for which $N = QD - 1$ is correctly determined by DBM_a. Therefore, the value of N_{cr} is $(Q + 1)D - 1$ (unless greater than the maximum value that may be stored in a word).

3. QUANTITATIVE ANALYSIS

In this section, the quantitative analysis of the method relative to the critical values of N associated with the divisors is presented. The results in Table 1 refers to odd divisors and in Table 2 to even divisors.

For the two most common word lengths $W = 32$ and $W = 64$, in the tables, are reported:

(i) L, the number of significant bits of the divisor;
(ii) N_{D}, the total number of divisors of length L;
(iii) AD_{32} and AD_{64}, the number of adverse divisors (AD) associated with a critical dividend of length 32 or 64 bits. The numbers AD_{32} are computed by an exhaustive search through all the possible divisors of length up to $L = 32$. The numbers AD_{64} are computed for all possible divisors up to $L = 32$ and for a significant subsample of divisors (2^{30} for odd, and $2^{30} - 1$ for even) from $L = 33$ to 64.

TABLE 1. Number of odd divisors associated with critical dividends as a function of L (upto $L = 32$) for $W = 32$ and 64

L	$N_{\text{D}} = 2^{L-2}$	AD_{32}	AD_{64}
2	1	0	0
3	2	1	1
4	4	0	0
5	8	4	5
6	16	8	7
7	32	16	18
8	64	21	19
9	128	47	57
10	256	86	90
11	512	165	179
12	1024	332	335
13	2048	623	656
14	4096	1264	1254
15	8192	2486	2523
16	16384	5116	5123
17	32768	10090	10314
18	65536	20275	20307
19	131072	40320	40582
20	262144	80325	80327
21	524288	161180	161116
22	1048576	321100	322130
23	2097152	643142	643379
24	4194304	1282189	1286690
25	8388608	2560165	2573619
26	16777216	5083457	5146048
27	33554432	10036907	10294620
28	67108864	19545436	20584633
29	134217728	37005336	41195018
30	268435456	65871664	82367318
31	536870912	104977250	164759665
32	1073741824	184204572	329480704

Let us call FSP_{32} and FSP_{64} the fraction of 32 and 64 bits dividends' number space, for divisors of length L, that are greater than or equal to critical dividends, averaged over those divisors:

$$\text{FSP}_W^{(L)} = \frac{1}{\text{AD}_W^{(L)}} \sum_{t=1}^{\text{AD}_W^{(L)}} \left(1 - \frac{N_{\text{cr}}(D_{tL})}{2^W}\right),$$

where D_{tL} is the tth divisor (odd or even) of length L that is associated with a critical dividend $N_{\text{cr}}(D_{tL})$. The values of

TABLE 2. Number of even divisors associated with critical dividends as a function of L (upto $L = 32$) for $W = 32$ and 64

L	$N_D = 2^{L-2} - 1$	AD_{32}	AD_{64}
3	1	0	0
4	3	1	1
5	7	1	1
6	15	5	6
7	31	13	13
8	63	29	31
9	127	50	50
10	255	97	107
11	511	183	197
12	1023	348	376
13	2047	680	711
14	4095	1303	1367
15	8191	2567	2621
16	16383	5053	5144
17	32767	10169	10267
18	65535	20259	20581
19	131071	40533	40888
20	262143	80843	81470
21	524287	161134	161797
22	1048575	322188	322913
23	2097151	642776	645043
24	4194303	1283839	1288422
25	8388607	2557818	2575112
26	16777215	5085153	5148731
27	33554431	10037650	10294779
28	67108863	19548892	20589399
29	134217727	37008932	41174032
30	268435455	65867093	82369050
31	536870911	104964226	164736368
32	1073741823	184245372	329496033

FSP_{32} and FSP_{64} are presented in Fig. 2 for odd divisors and Fig. 3 for even divisors as a function of L.

From our extensive analysis, we are convinced that an analytic determination of AD_{32} and AD_{64} does not exist.

With respect to Table 1, L in Table 2 begins at 3 because there is no even divisor of $L = 2$ which is not a power of 2. As well our analysis finds that 10 is a divisor *not* associated with critical dividends of size 32 or 64 bits. This means that DBM_a may be used directly to convert from binary to decimal (when the divisor is 10, for $W = 32$, $J = 3435973837$ whereas for $W = 64$, $J = 14757395258967641293$).

We note that the major part of all divisors is not associated with critical dividends and that the fraction FSP of dividend space occupied by dividends $N \geq N_{cr}$ seldom exceeds 25% averaged over the critical divisors of a given size in both 32 and 64 bits dividend word sizes (Figs. 2 and 3). Similar values of FSP have been obtained for $W = 64$ and a large subsample of divisors of length $32 < L \leq 64$.

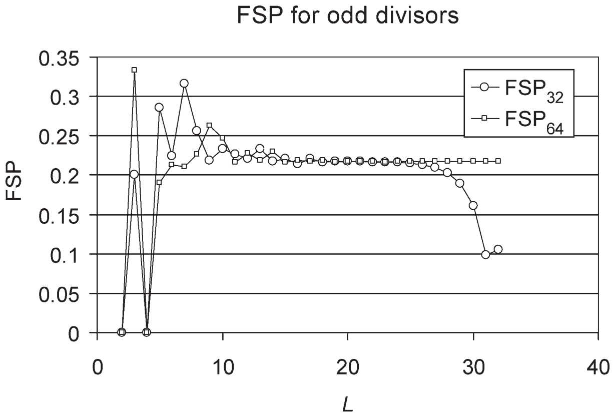


FIGURE 2. The fraction of 32 and 64-bit dividends greater than N_{cr} for odd divisors of length L that lead to critical dividends, averaged over those divisors.

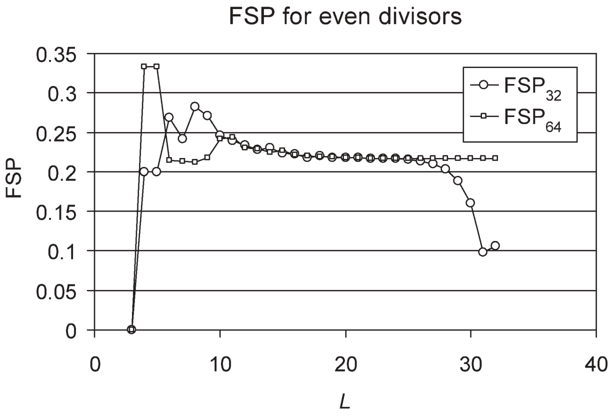


FIGURE 3. The fraction of 32 and 64-bit dividends greater than N_{cr} for even divisors of length L that lead to critical dividends, averaged over those divisors.

By comparing AD_{32} with AD_{64} in Table 1, one can note that clearly some odd divisors are associated with critical dividends when the word size is 32 and not when 64, and vice versa. The same is true for even divisors, although not evident from comparing AD_{32} with AD_{64} because $AD_{64} \geq AD_{32}$ for all L values in Table 2. Also, the fraction of dividend space occupied by critical dividends becomes very small as the divisor length approaches the word size, as in FSP_{32}. The same behaviour was observed for $W = 64$ on the subsample of divisors examined.

For roughly half of the divisors in which the fractional part of FC/D is greater than or equal to 0.5, DBM_a is the same algorithm as the one in [10, Section 16.9] for these cases because for divisors D having a fractional part of FC/D greater than or equal to 0.5 there is no critical dividend as proven in the appendix.

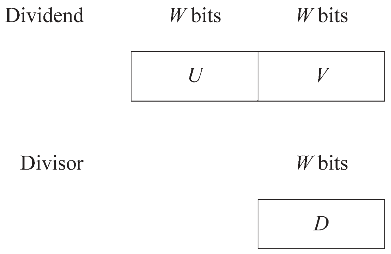
Differences arise when the fractional part of FC/D is less than 0.5. In these cases (statistically 50% of divisors), the algorithm in [10, Section 16.9] requires summing 1 to the dividend before multiplication by I and right shifting. In our framework, the introduction of a critical dividend (and a multiplication by the multiplicative inverse J) allows the use of multiplication and right shift for all these divisors not having a corresponding critical dividend. From Tables 1 and 2, we found (for both 32 and 64 bits environments) that ~20% of all divisors have critical dividends. The remaining 30% do not have critical dividends, allowing DBM to avoid checking for the decrement or setting the LSB to 0. Figure 4 makes the percentage distribution clear.

Considering the 50% of cases having a fractional part of FC/D greater than or equal to 0.5 for which no critical dividend is possible as shown in the appendix along with the 30% of divisors leading to a fractional part of FC/D less than 0.5 and no N_{cr}, we have that in 80% of all cases DBM requires a multiplication and a shift only. For the remaining 20% of all divisors, we may zero the LSB of the dividend in case of even divisors (10%), also in these cases saving a sum w.r.t. the algorithm in [10, Section 16.9]; for the remaining odd divisors (10%), the DBM requires a comparison and a *possible* decrement.

The decrement is required only for a fraction of the dividend space (i.e. dividend greater than or equal to N_{cr}); averaging FSP_{32} for all odd divisors associated with critical dividends one can find that the fraction of their space being greater than or equal to N_{cr} is ~13%. Thus, the dividend decrement is required only in $10\% \times 13\% = 1.3\%$ of all possible cases. In practice, the algorithm in [10, Section 16.9] requires summing 1 to the dividend in 50% of all possible cases, but if $2^W - 1$ is a possible dividend, then also a comparison for overflow (after the sum) is required.

4. DIVIDING UNSIGNED DOUBLE WORD BY UNSIGNED WORD

The DBM is designed to divide an unsigned word by an unsigned word. There may be situations in which it is necessary to divide an unsigned double word by an unsigned word D. The aim here is to show that it is possible to perform this division using only multiplications, shifts, compare-branches and sums, besides DBM. The first idea is to apply DBM (using the divisor D) to the upper and to the lower words of the dividend computing the quotients and the remainders:



$$Q_U = \mathrm{DBM}(U); \quad R_U = U - Q_U \times D, \qquad (17)$$
$$Q_V = \mathrm{DBM}(V); \quad R_V = V - Q_V \times D.$$

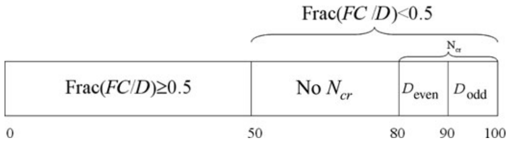


FIGURE 4. Percentage of divisors and their properties.

Thus, $Q_U \times 2^W$ and Q_V are the first contributions to the quotient. Another contribution to the quotient is obtained dividing $R_U \times 2^W$ by D, but this involves a double word dividend, in fact in

$$\left\lfloor \frac{R_U 2^W}{D} \right\rfloor \qquad (18)$$

the numerator is a double word.

To circumvent the problem, one may compute at the compile time the value

$$T = \left\lfloor \frac{2^W}{D} \right\rfloor \qquad (19)$$

and then perform the multiplication $Q_U{}' = R_U \times T$ to compute a first approximation to the quotient in (18) to be added to the final quotient.

Given that

$$R_U T = R_U \left\lfloor \frac{2^W}{D} \right\rfloor \leq \left\lfloor \frac{R_U 2^W}{D} \right\rfloor \qquad (20)$$

we are ensured that the remainder is non-negative:

$$R_U \times 2^W - R_U \times T \times D \geq 0. \qquad (21)$$

The problem that arises is that the new value on which to perform a division is

$$R'_U = \mathrm{MSW}[R_U \times 2^W + R_V - Q'_U \times D] \qquad (22)$$

(where MSW[X] is the upper word of the double word X). The value R'_U may be greater than 0, requiring an iteration of this procedure while also taking into account the value

$$V' = \mathrm{LSW}[R_U \times 2^W + R_V - Q'_U \times D], \qquad (23)$$

i.e. the lower part of the double word that still needs division (LSW[X] is the lower word of the double word X).

To accelerate the iterative convergence, it is possible to increase the precision in the computation of $Q_U{}'$ taking into account the fractional part in the division in (19). The new value of T to be used is then

$$T = \left\lfloor \frac{2^{2W}}{D} \right\rfloor. \tag{24}$$

Note that this integer division involves 1 bit more than a double word in the numerator, but it is possible to perform this division at compile time using assembly language. To simplify the following analysis, we put aside for the moment R_V.

Then it is possible to compute $R_U \times T$, but the new contribution Q'_U to the final quotient requires to divide this value by 2^W:

$$Q'_U = \left\lfloor \frac{R_U \lfloor 2^{2W}/D \rfloor}{2^W} \right\rfloor \tag{25}$$

The new value of T proposed in (24) allows us to determine a limit for the remainder

$$R_U 2^W - DQ'_U = R_U 2^W - D\left\lfloor \frac{R_U \lfloor 2^{2W}/D \rfloor}{2^W} \right\rfloor. \tag{26}$$

Remember that for $a \geq 0$, $b > 0$, $a/b - 1 < \lfloor a/b \rfloor \leq a/b$.

We may use these inequalities to give limits to the value in (26).

First,

$$\frac{2^{2W}}{D} - 1 < \left\lfloor \frac{2^{2W}}{D} \right\rfloor \leq \frac{2^{2W}}{D},$$

thus

$$R_U\left(\frac{2^{2W}}{D} - 1\right) < R_U\left\lfloor \frac{2^{2W}}{D} \right\rfloor \leq R_U \frac{2^{2W}}{D}, \tag{27}$$

Second,

$$\frac{R_U \lfloor 2^{2W}/D \rfloor}{2^W} - 1 < \left\lfloor \frac{R_U \lfloor 2^{2W}/D \rfloor}{2^W} \right\rfloor \leq \frac{R_U \lfloor 2^{2W}/D \rfloor}{2^W}$$

and, from (27), we may remove the 'floor operator' from the leftmost and rightmost parts

$$\frac{R_U(2^{2W}/D - 1)}{2^W} - 1 < \left\lfloor \frac{R_U \lfloor 2^{2W}/D \rfloor}{2^W} \right\rfloor \leq \frac{R_U 2^{2W}/D}{2^W},$$

$$\frac{R_U 2^{2W} - R_U D - 2^W D}{2^W D} < \left\lfloor \frac{R_U \lfloor 2^{2W}/D \rfloor}{2^W} \right\rfloor \leq \frac{R_U 2^{2W}}{2^W D}.$$

Multiplying all the members of the inequalities by $-D$ and adding $R_U \times 2^W$, we obtain

$$\frac{R_U D}{2^W} + D > R_U 2^W - D\left\lfloor \frac{R_U \lfloor 2^{2W}/D \rfloor}{2^W} \right\rfloor \geq 0.$$

However, given that $2^W > R_U$, then

$$2D > R_U 2^W - D\left\lfloor \frac{R_U \lfloor 2^{2W}/D \rfloor}{2^W} \right\rfloor \geq 0. \tag{28}$$

The inequalities in (28) explain us that the remainder of the approximate division of $R_U \times 2^W$ by D made using the factor T is non-negative and is less than $2 \times D$.

From (28), an algorithm to compute the division is shown below (variables have the same meaning as in the previous, discussion):

```
$Q_U = \mathrm{DBM}(U)$;
$Q_V = \mathrm{DBM}(V)$;
$R_U = U - Q_U \times D$;
$R_V = V - Q_V \times D$;
 $Q = \mathrm{UpWrd}(R_U \times T)$; //Consider the upper word of the result only
REM = $R_U \times 2^W + R_V - Q \times D$; // Note: $R_U \times 2^W + R_V$ is a double word
// made composing the words $R_U$ and $R_V$
// thus no sum is needed!
// REM is less than $3 \times D$
If REM >=2 × $D$ then $Q$=$Q$+2; REM=REM − 2 × $D$;
Else if REM >= $D$ then $Q$=$Q$+1; REM=REM − $D$;
$Q$=$Q$+$Q_U \times 2^W$+$Q_V$; // Note: $Q_U \times 2^W + Q_V$ is a double word
// made composing the words $Q_U$ and $Q_V$
// thus no sum is needed!
```

This algorithm has a finite number of steps to produce the result.

The first consideration is that the multiplication $R_U \times T$ involves an unsigned word and an unsigned double word: this multiplication is performed with two multiplications of single words and a sum of two single words since only the upper part is needed. Thus, the cost of the previous algorithm is seven unsigned word multiplications, 14 single word sums (subtractions are counted as sums), six single word comparison-branches and two shifts. A by-product of these operations is the final remainder. An extensive random simulation of all possible widths of dividends and of divisors has shown that the final remainder requires increasing Q by 1 about half the time; increasing Q by 2 is very rare and occurs only for large dividends and large divisors (where R_U is of the same order as 2^W).

The second consideration is that the presented algorithm makes no assumptions on dividend and divisor. Thus, the dividend may be any possible value of an unsigned double word and the quotient may consequently be a double word.

The third observation regards the computation of T and I used in DBM. As can be noted by comparing (1) and (26), I is an intermediate result in the determination of T. This simplifies the work of the compiler in case an unsigned double word must be divided by an unsigned single word and T is thus required.

5. CONCLUSIONS

We have developed an efficient method for accelerating integer division when the divisor is known at compile time. We believe that our derivation and the associated algorithms make very clear the nature of this process.

The introduction of the critical dividend has provided a way to make very explicit the limit of the straightforward calculation of the multiplicative inverse for divisors. Either method for calculating N_{cr} is valid. We used both methods in parallel and verified that the same value resulted every time. The division method is more direct because it does not require the comparison between Q in calculating the test dividend and the Q_{r} from DBM_a. The multiplication method is a trial-and-error method and does not require any division.

We find that our adverse divisors correspond exactly to the 'bad' divisors in [9], but the use of our critical dividend leads to a more efficient division algorithm.

Through our numerical determination of the dividend space in which critical dividends can appear, we find that the major part of ALL divisors (both even and odd) do not involve critical dividends (80%). While the percentage of divisors associated with critical dividends is much greater than 20% for certain L values, the 20% result comes from weighting the single L percentages with the available number of divisors for $L = 2$ (3 for even) up to $L = 32$. For odd divisors having a critical dividend (10% of all divisors), the fraction of the corresponding dividend space subject to the operation of decrementing is 13%. These results are presented for divisors up to 32 bits and dividends of size 32 and 64 bits in Tables 1 and 2 and Figs. 2 and 3. An advantage of our approach is the way the single-word dividend method can be integrated into the presented algorithm for the division of a double-word dividend by a single-word divisor providing the remainder directly and without imposing limits on the length of the quotient.

It is not necessary to extend the analysis to signed division where all magnitudes are less than or equal to 2^{W-1} and always less than eventual critical dividends. For signed division, we suggest the use of the same relations as given in [12, Section 3].

One last consideration regards the use of this algorithm. Obviously, it is applicable when the divisor is a constant known at compile time. Also, when the compiler may deduce that a division operation is executed *many times* (e.g. in a loop) and the divisor does not change (but is known only at run-time), then it may generate the code for computing the constants needed by DBM (i.e. J and, if necessary, N_{cr}), and insert the DBM code in the loop. Here, *many times* means an amount of times that justifies the effort of computing the constants J and N_{cr}.

ACKNOWLEDGEMENTS

The authors thank Professors P.C. Giolito, A.R. Meo and P. Montuschi for useful advice, and Dr S. Rabellino for the interest and suggestions in this work. The authors also thank A. Fog and T. Mathisen for the information in [10] and the algorithm we commented in our paper. We appreciate the comments of the anonymous reviewers, which helped to make our paper more readable, comprehensible and integrated.

FUNDING

This work was supported by local research funds of the Università degli Studi di Torino.

REFERENCES

[1] Sites, R.L. (Ed.) (1992) *Alpha Architecture Reference Manual*. Digital Press. ISBN 1-55558-0 98-X.

[2] Intel Corporation (2002) *Intel Itanium Architecture Software Developer's Manual, vol. 3: Instruction Set Reference*, Rev. 2.1.

[3] IBM (1994) *The PowerPC^{TM} Architecture: A Specification for a New Family of RISC Processors* (2 edn). Morgan Kaufmann. ISBN 1-55860-316-6.

[4] Yeagar, K.C. (1996) The MIPS R10000 Superscalar Microprocessor. *IEEE Micro.*, **16**, 28–40.

[5] Cavanaugh, J. (1984) *Digital Computer Arithmetic Design and Implementation*. McGraw-Hill. ISBN 0-07-010282-1.

[6] Granlund, T. and Montgomery, P.L. (1994) Division by Invariant Integers Using Multiplication. *ACM SIGPLAN*, **vol. 29**, pp. 61–72.

[7] Mantovani, G., Werbrouck, A. and Zerbone, L. (1997) Division By Multiplication. Technical Report RT44/97. Dipartimento di Informatica, Università degli Studi di Torino.

[8] Srinivasan, P. and Petry, F.E. (1994) Constant-division algorithms. *IEE Proc., Comput. Digit. Tech.*, **141**, 334–340.

[9] Henry Jr, S.W, (2002) *Hacker's Delight*. Addison-Wesley.

[10] Fog, A. (2007) *Optimizing subroutines in assembly language*. File optimizing_assembly.pdf, available at web site http://www.agner.org

[11] Jacobsohn, D.H. (1973) A combinatoric division algorithm for fixed-integer divisors. *IEEE Trans. Comput.*, **C-22**, 608–610.

[12] Muller, J.-M., Tisserand, A., de Dinechin, B. and Monat, C. (2005) Division by Constant for the ST100 DSP Microprocessor. In Montuschi, P., and Schwarz, E. (eds), *Proc. 17th IEEE Symp. Computer Arithmetic* (ARITH'05), pp. 124–130. IEEE Computer Society, Cape Cod, MA, USA.

6. APPENDIX

In this section, we present the solutions for integer division using the multiplication proposed in [10, Section 16.9] and prove the correctness of these solutions (which we have not found in the literature) using our framework.

We use $I = \lfloor FC/D \rfloor$ which in [10, Section 16.9] appears as $f = FC/D$, including the fractional part of this quotient and examine this value. Let us use the following notation: $f = I{\cdot}h$, where I is the integral part of the quotient (as in our case) and h is the fractional part. The symbol r has the same meaning as our $W + L - 1$; SHR means right shift; *result* is the result of the integer division.

Case h < 0.5 (rounding bit = 0)

result = $((N + 1)*I)$ SHR r

Proof Given that in this case $h < 0.5$, and $I{\cdot}h = FC/D = 2^{W+L-1}/D$,

$$DI + \frac{D}{2} > DI + Dh = FC > DI > FC - \frac{D}{2},$$
$$0 \le RI \le DI - I.$$

From $N = QD + R$ and the above inequalities,

$$QDI \le NI \le QDI + DI - I,$$
$$Q + \frac{(2I - QD)}{2FC} < \frac{(N + 1)I}{FC} < Q + \frac{DI}{FC} < Q + 1.$$

From $DI < FC$, we have that the value $(N + 1)I/(FC)$ is less than $Q + 1$, and thus the integer part is less than or equal to Q.

We have previously shown that $2^{W-1} \le I$, and thus $2^W \le 2I$. Also, $QD \le N < 2^W \le 2I$, then the value computed by the algorithm is greater than Q plus some fractional part.

From both inequalities, we deduce that the integer part of $(N + 1)I/(FC)$ is equal to Q. □

Case h ≥ 0.5 (rounding bit = 1)

This method is the same as ours and the interest in the proof is that a critical dividend cannot exist.

result = $(N*(I + 1))$ SHR $r = (N*J)$ SHR r

Proof

$$FC = DI + Dh \ge DI + \frac{D}{2},$$
$$FC - \frac{D}{2} \ge DI, \quad FC + \frac{D}{2} \ge D(I + 1),$$

From $N = QD + R$

$$N(I + 1) = QD(I + 1) + R(I + 1) = QDI + QD + R(I + 1),$$
$$0 \le R(I + 1) \le (D - 1)(I + 1),$$
$$QDI + QD \le N(I + 1) \le Q\left(FC - \frac{D}{2}\right) + QD + (D - 1)(I + 1).$$

From (2), we have $FC - D < DI$; also $2^{W-1} \le I$, that is $-(I + 1) \le -(2^{W-1} + 1)$. Thus,

$$Q(FC - D) + QD < N(I + 1) \le QFC + \frac{QD}{2} + D(I + 1) - (I + 1),$$
$$Q < \frac{N(I + 1)}{FC} \le Q + 1 - \frac{2^W + 2 - QD - D}{2FC}.$$

Let us examine the expression $2^W + 2 - QD - D$.

We may write $2^W + 2 - QD - R + R - D = 2^W + 2 - N + R - D$.

We have also seen that the case of a critical dividend is possible only in case $R = D - 1$. Now, substitute this value for R and obtain:

$$2^W + 2 - N + R - D = 2^W + 2 - N + D - 1 - D$$
$$= 2^W + 2 - N - 1 = 2^W + 1 - N.$$

But $N < 2^W$, and thus $2^W + 1 - N > 0$. This means that

$$Q < \frac{N(I + 1)}{FC} < Q + 1$$

implying that the only possible value for the floor integer part of $N(I + 1)/(FC)$ is Q. This also shows analytically that in the case $h \ge 0.5$ no critical dividends are possible. □