Journal of Universal Computer Science, vol. 1, no. 3 (1995), 67-82
submitted: 13/3/95, accepted: 20/3/95, appeared: 28/3/95©Springer Pub. Co.

Special Cases of Division

R W Doran

(The University of Auckland, New Zealand. bob@cs.auckland.ac.nz)

Abstract: This surveys algorithms and circuits for integer division in special cases. These include division by constants, small divisors, exact divisors, and cases where the divisor and the number base have a special relationship. The related operation of remainder is also covered. Various prior techniques are treated in a common framework. Worked examples are provided together with examples of practical application.
Category: B2.0 Arithmetic and Logic Circuits

1 Introduction

Division, although known in theory to be capable of O(log n) solution [Beame, Cooke and Hoover 1986], is difficult to implement with high performance in practice. However, there are many special cases where division is much easier, so that fast algorithms may be used. Many "tricks" have been discovered over the years. Some of these are not well known and there are others that are familiar to practitioners but have not been described in the literature that is easy to access. The purpose of this paper is to survey the special cases of division and describe them in a uniform manner. We will consider only non-negative integers, extended to fixed-point number representation in some cases.

1.1 Notation

To describe division we will use the notation:

D	dividend	d	divisor
q	quotient	r	remainder

where all the values involved are non-negative integers, with d > 0.

In general, division is the process that, given D and d, finds q and r so that D = q*d + r where $0 \le r < d$. We will sometimes express q as D div d and r as D mod d (r being often referred to as the residue modulo d - remainder and modulo are sometimes defined differently for signed numbers, but are the same for the non-negative integers with which we are concerned here). Usually, the term *division* is restricted to finding q, and the corresponding process of finding r as *remaindering.*

We will assume that integers are presented, using a positive integer base b, as n-digit vectors. In particular we have **D**, **d**, **q**, **r**, where, for example:

$$D = \mathbf{D}_{n-1}\, b^{n-1} + \mathbf{D}_{n-2}\, b^{n-2} + \mathbf{.......} + \mathbf{D}_0 b^0$$

The values $\mathbf{D}_i$ are the digits in the base b representation. $0 \le \mathbf{D}_i < b$. The digits are uniquely determined.

We will often use base 10 in examples although most circuits will use binary representation in practice.

1.2 Relationship between div and mod

The operations of division and remaindering are closely related and are reducible to each other. That the remainder can be found after division is obvious, for:

$$r = D \bmod d = D - (D \text{ div } d)*d = D - q*d$$

Knowledge of the remainder can reduce division to the special case where the dividend is an exact multiple of the divisor, for D - (D mod d) = q*d. This can simplify the process of division in some circumstances to be covered below. However, the ability to perform remaindering can allow the quotient to be derived without further division, but the process depends on the representation of the numbers and the standard algorithm for division.

Write $D^{(i)} = \mathbf{D}_{n-1}\, b^{n-1-i} + \mathbf{D}_{n-2}\, b^{n-2-i} + \ldots\ldots + \mathbf{D}_i\, b^0$ and $R^{(i)} = D^{(i)} \bmod d$. $D^{(i)}$ is the leading n-i digits of D regarded as an integer. The $R^{(i)}$ are called "partial remainders". The $D^{(i)}$ are each represented as a subrange of the digits **D**. It is possible from **D** to quickly make enough copies of its digits so as to represent all $D^{(i)}$ simultaneously (by *quickly*, we mean logarithmically in terms of time or levels of logic in a circuit). Given the $D^{(i)}$, if remaindering can can be performed quickly then we can apply it to all $D^{(i)}$ in parallel and so obtain all the $R^{(i)}$ quickly.

The standard process of division is to produce the quotient digits $\mathbf{q}_i$ in order, commencing with the high order $\mathbf{q}_{n-1}$. At step i (i from n-1 down to 0) we divide d into the "concatenation" of $R^{(i+1)}$ with the next digit $\mathbf{D}_i$ to find $\mathbf{q}_i$ and $R^{(i)}$. For example, in the following "trace" of standard long division the partial remainders are picked out in bold.

```
        0 2 2 6
      ---------
2 5 | 5 6 7 3
      0
      -
      5 6
      5 0
      ---
        6 7
        5 0
        ---
        1 7 3
        1 5 0
          ---
          2 3
```

At each step we have to perform the division expressed by $R^{(i+1)}*b+\mathbf{D}_i = d*\mathbf{q}_i + R^{(i)}$ (assume that $R^{(n)} = 0$). As we can find the $R^{(i)}$ in parallel quickly we can then determine the $\mathbf{q}_i$ by dividing each $R^{(i+1)}*b+\mathbf{D}_i$ by d.

Thus the standard process allows us to use remaindering to reduce general division to steps that involve division that produces a small quotient (the $\mathbf{q}_i$ are less than b) - this is the special case that we will cover first, below. However, here the situation is even more special. From above, we have $d*\mathbf{q}_i = R^{(i+1)}*b + \mathbf{D}_i - R^{(i)}$ which can be calculated quickly. ($R^{(i+1)}*b + \mathbf{D}_i$ involves no calculation, it is merely notation for concatenation. The subtraction of $R^{(i)}$ may be performed quickly in a borrow lookahead circuit.)

We now have division that is *exact* and with a *small quotient*. Because $\mathbf{q}_i$ is in the range from 0 to b-1, $\mathbf{q}_i$ may be deduced quickly from knowledge of the constant multiples of d in the range from 0 to b-1.

Example:

D = 5673, d = 25, **D** = (5,6,7,3)
$(D^{(3)},D^{(2)},D^{(1)},D^{(0)})$ = (5,56,567,5673), $(R^{(3)},R^{(2)},R^{(1)},R^{(0)})$ = (5,6,17,23),
d*$\mathbf{q}$ =(0,50,50,150). {calculated as d*$\mathbf{q}_i$ =$R^{(i+1)}$*10 +$\mathbf{D}_i$ - $R^{(i)}$ }
$\mathbf{q}$ = (0,2,2,6) {because 25*0 = 0, 25*1 = 25, 25*2 = 50 etc.)

In even more-special cases the process of finding the quotient digits is further simplified. For example, because $\mathbf{q}_i$ = 0 iff $R^{(i+1)}$*b+$\mathbf{D}_i$ = $R^{(i)}$, in the case of binary division $\mathbf{q}_i$ is fully determined by this comparison.[1]

In summary, if the values $R^{(i)}$ are known it is possible to determine the $\mathbf{q}_i$ quickly in parallel. We will see situations where it is much easier to find the partial remainders than to perform division directly, which is why we are treating special cases of both division and remaindering. The general process of deducing quotient from remainders in parallel is illustrated in [Fig. 1] for n = 8 (this is trivial but it is shown as it will be a component of later circuits, although simplified further - note that the divisor d is an assumed input to all subcircuits).

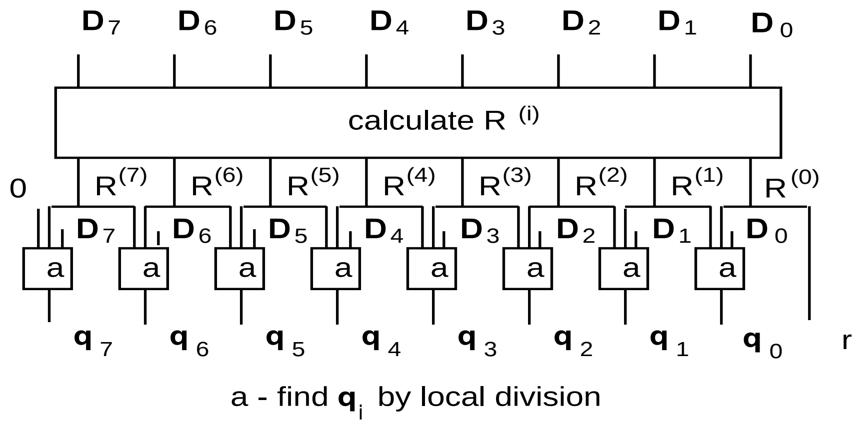


Figure 1: Derivation of quotient from remainders

2 Small quotient

There are sometimes circumstances where it is known that the quotient is small, so that we can find it by using case analysis. Because D = d*q + r, for each possible q' we know that q = q' iff d*q' $\leq$ D < d*(q'+1).

One situation arose in the standard division process above where we needed to find each $\mathbf{q}_i$ in turn from $R^{(i+1)}$ *b+$\mathbf{D}_i$ = d*$\mathbf{q}_i$ + $R^{(i)}$, i.e. by dividing d into $R^{(i+1)}$ *b+$\mathbf{D}_i$ to find the quotient $\mathbf{q}_i$ that we know is < b. Using the same data as above:

Example:

$R^{(i+1)}$*b +$\mathbf{D}_i$ produces d*$\mathbf{q}_i$ + $R^{(i)}$ as (5,56,67,173), which are, by inspection, in the ranges (0-24,50-74,50-74,150-174), so q = (0,2,2,6).

---

[1] In [Beame, Cooke and Hoover 1986] this reduction is credited to [Alt and Blum 1983], but it appears to have been well understood by practitioners, and is used in, for example, [Cocke et. al. 1970].

Another situation arises in arithmetic modulo d where x < d and we want to form (x * y) mod d, where y < b, the number base, which is small. Setting D = x*y, and q = D div d, we know that q < b. Because there are only a limited number of possibilities for the quotient, it can be found first and used to calculate the remainder which has a much wider range of values.

Given that the quotient is small, its value may be estimated in many cases by looking at the first few digits of the values i*d, for the possible values of i, rather than by performing full comparisons.

Example:

> d =567, q < 5.
> From the first digits of d we can make the following definite decisions.
>
> > D < 0500 -> q = 0,
> > 0600 $\leq$ D <1100 -> q = 1,
> > 1200 $\leq$ D < 1700 -> q = 2,
> > 1800 $\leq$ D < 2200 -> q = 3,
> > D $\geq$2300 -> q = 4.
>
> So, if D = 1939, we know that, q=3 because D has leading digits 19 and so r = D - q*d = 238. However, if D commences with 17 we will have to consider more than two digits of D in order to distinguish between q=2 and q=3.

Full comparisons will be needed to distinguish some cases unless d has other special properties, or unless complete accuracy in determining q is not required.

3 Constant divisor

3.1 Multiplication by the reciprocal of the divisor

Division by d (_0) may always be performed by multiplication by its reciprocal 1/d.

Example:

> to divide D = 99866 by d = 167 find 1/d ~ 0.005988.
> 99866/167 = 99866*0.005988 = 597.99 = 598 rounded.

This is the basis for division in computers such as the Cray 1 series where there is no division instruction. Rather, an instruction is provided that gives an approximation to the reciprocal. Division is performed using software to refine the reciprocal approximation to full precision and to multiply by the dividend [Iliffe 1982]. It is not a popular method for implementing division in hardware because repetitive methods are more facile. However, if there is a need to implement division by a particular constant then the reciprocal may be calculated once and for all in advance.

Perhaps the most common use of division by a constant is in conversion between number bases. There are two approaches, one which generates the least significant digit first as the remainder from division by a small constant - we will encounter that later. The other is to divide by a large constant and generate the most significant digit first. Suppose it is required to convert an n-bit binary number N into a BCD decimal representation providing always the maximum number of digits m, where $10^m \geq 2^n$. Firstly, N can be divided by 10^m (in binary) then the quotient multiplied successively by 10 (in binary, multiplication by a small constant can, of course, be performed by a sequence of additions and shifts), collecting 4-bit decimal BCD digits as they appear to the left of the binary point. Division by 10^m can be performed by multiplying by $1/10^m$ (kept with sufficient precision to convert the largest binary integer).

Example:

Assume n=6 and m=2.
$d=10^2$ = 1100100 and $1/d = 1/10^2 = 0.10100111*2^{-6}$.
To convert N=110001 (49) to BCD, first form:
$N/d = N*(1/d) = 110001*0.10100111*2^{-6} = 0.01111110$
Now, 1. N*(1/d) * 1010 (ten) = 0100.1111, i.e. 0100 (4) + 0.1111
2. 0.1111*1010 (ten) = 1001.0110, i.e. 1001 (9) plus an insignificant remainder. So $((110001 / 10^2)^{*}10)*10 = (4*10^1 + 9)$
i.e. $110001 = 4*10^1 + 9$.

3.2 Direct calculation of the remainder for constant divisor

Calculation of D mod d, where d is a constant, can be performed by calculating D - (D*(1/d))*d, but this requires two multiplications. An alternative method is based on stored constants and properties of the modulus operation.

$$\begin{aligned} D \bmod d &= (\mathbf{D}_{n-1} b^{n-1} + \mathbf{D}_{n-2} b^{n-2} + \mathbf{....} + \mathbf{D}_0 b^0) \bmod d \\ &= ((\mathbf{D}_{n-1} b^{n-1}) \bmod d + (\mathbf{D}_{n-2} b^{n-2}) \bmod d + \ + (\mathbf{D}_0 b^0) \bmod d) \bmod d \\ &= E \bmod d, \\ \text{where } E &= (\mathbf{D}_{n-1} b^{n-1}) \bmod d + (\mathbf{D}_{n-2} b^{n-2}) \bmod d + \mathbf{....} + (\mathbf{D}_0 b^0) \bmod d \end{aligned}$$

The constants $(k\ b^i) \bmod d$ $(0 \leq k < b)$ can be stored in n tables with b entries of n digits, $(\mathbf{D}_i\ b^i) \bmod d$ selected by table look up, and then E found as the sum. The addition can be performed in software, or, in hardware, in logarithmic time using a tree of adders.

E is in the limited range from 0 to n*d, so we can now use techniques for small quotients to determine k so that $kd \leq E < (k+1)d$. D mod d, = E mod d, is then E - kd. This can again be performed in software or, if speed is of essence, in hardware by comparing E in parallel with all the constants i*d.

[Fig. 2] shows an example for the case of n=8. Note that the additions, other than the last, may be carry-save so that the adder tree has similar cost and complexity to a multiplier.

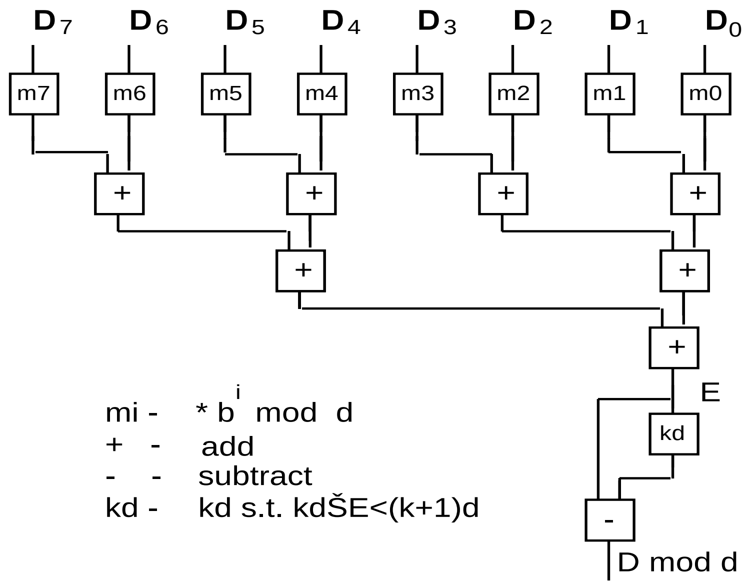


Figure 2: Calculation of D mod d when d is constant

A direct application of remaindering is range scaling for fixed-point numbers. Many processes for calculating elementary functions require that the argument be within a limited range, eg. [0 : $\pi/2$] for sin. To calculate sin(x) it is necessary to reduce the range of x by determining x mod $\pi/2$. It is possible to make the final "small quotient" step fast, using only a small table look-up, if, as is often the case, the actual range of convergence is somewhat wider than the target range [Daumas et. al. 1994]. That is, we can tolerate an error in finding kd when x is close to a multiple of d as the effect is to slightly increase the range, in this case from [0 : $\pi/2$] to [0 : $\pi/2$ + delta].

4 Small divisor

Small divisors are important in both theory and practice. If small is taken to mean of length O(log n), then there are O(n^k) different numbers representable. Dealing with a variable integer of length O(log n) is thus the same as considering O(n^k) different cases. A hardware selection from O(n^k) results may be made in time O(log n). Hence, operating on small variables is equivalent, in speed, to operating with constants, plus time O(log n) for selection.

One use of small variables is in residue arithmetic [Szabo and Tanaka 1967]. In residue arithmetic, O(n/log n) distinct prime numbers m_i each of length O(log n) are chosen. A number X of length O(n) is then represented by the small numbers x_i where x_i = X mod m_i. The advantage of this system is that (X *op* Y) mod M is performed, for many *op*, as (x_i *op* y_i) mod m_i in parallel. Unfortunately, division is, in general, not in this category. Regardless of the benefit of this approach it leads to interest in operations on numbers of length O(log n).

For example (from [Beame, Cooke and Hoover 1986]), to calculate (D mod d) mod m where D is n digits and m and d are O(log n) digits and d is variable. Tables of the constants $\mathbf{D}_i$ b^i mod d for every possible $\mathbf{D}_i$ and d have size b*n^k for each i. Reference to such a table given $\mathbf{D}_i$ and d takes time log $(b*n^k)$ = O(log n). Thus, the direct calculation approach for determining D mod d given above for constant d may be used with variable small d to also give a O(log n) algorithm.

Under the same conditions, (D div d) mod m may be found using the equivalence between mod and div. For (D^{-1}) mod m (where D^{-1} is defined as the integer < m such that $D*D^{-1} = 1$ mod m), we may compute y = D mod m and then look up y^{-1} in a table of size n^k.

A related trick is used in practice in ordinary full division or calculation of inverse. To start an iterative process going an approximation to the reciprocal of the divisor is needed. This may be looked up in a table using the first few bits of the divisor itself. This is used, for example, in the reciprocal approximation instruction in the Cray computers mentioned above [Iliffe 1982].

5 Small constant divisor

When the divisor is both constant and small there are further possibilites for simplifying division. This fortuitous combination does occur in practice. A hardware application is with interleaved memory banks. If there are k banks then an address A may be located in bank *A mod k* at address *A div k*. Similar calculations are required at the software level if the size of data items packed into memory differs from the memory word size. Another application is in conversion between number bases.

There are two directions that may be taken, one based on remaindering and the other on multiplication by reciprocals.

5.1 Division following derivation of remainders

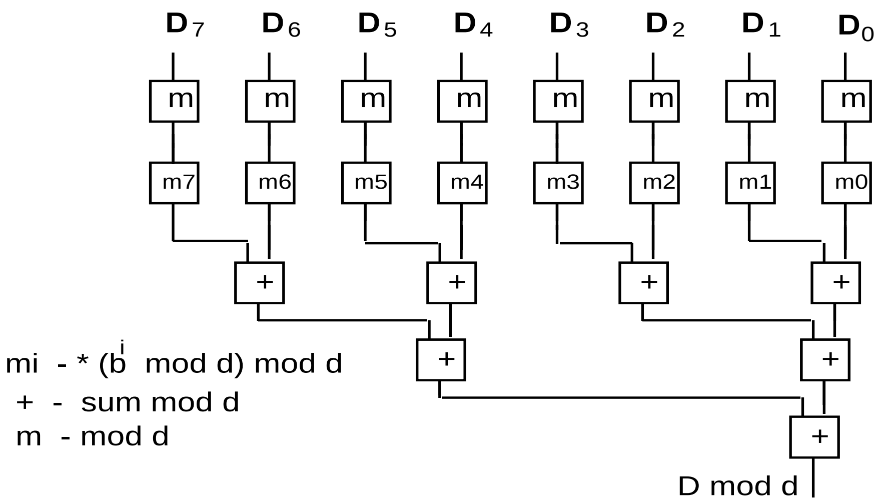


Figure 3. Remainder calculation for small divisor

Here one applies the same technique as before for constant divisors, but, because the divisor is also small, it is possible for all additions to be perfromed modulo the divisor, keeping the intermediate values small. The obvious circuit, [Fig. 3], calculates $\mathbf{D}_i$ mod d for each digit, then finds ($\mathbf{D}_i$ mod d)*(2^i mod d) mod d. At each level of the adder tree, addition mod d is performed.

Example:

D=93670341, d = 7, 10^i mod d = (3, 1, 5, 4, 6, 2, 3,1)
$\mathbf{D}_i$ * 10^i mod d = (6, 3, 2, 0, 0, 6, 5, 1)
-> (2, 2, 6, 6)
-> (4, 5)
-> (2)

Because the partial remainders are small, it is now reasonable to calculate all in parallel and hence deduce the complete quotient. The above approach has to be modified because the intermediate steps in calculating D mod d (= $R^{(0)}$) do not help in finding the other partial remainders $R^{(i)}$. A clever procedure [Cocke et. al. 1970] is to omit the *b^i mod d step and do the reduction addition as A*2^j+B mod d steps, thus spreading the "*b^i " operation over multiple stages. This gives the approach shown for n=8 in [Fig. 4] , refining the circuit of [Fig. 1].

Example:

D=93670341, d = 7, m = *mod 7*
p1 = *(A*3+B) mod 7*, p2 = *(A*2+B) mod 7*, p3 = *(A*6 + B) mod 7*,
p4 = *(A*4+B) mod 7*, c = *(A*10+B) div 7*.
(9,3,6,7,0,3,4,1) -> (2,3,6,0,0,3,4,1) -> (2,2,6,4,0,3,4,6) ->
(2,2,5,1,0,3,1,5) -> (2,2,5,1,3,5,5,2)
So $R^{(i)}$ = (0,2,2,5,1,3,5,5,2). $R^{(i+1)}$ *10 + $\mathbf{D}_i$ = (09,23,26,57,10,33,54,51)
q = ($R^{(i+1)}$ *10 + $\mathbf{D}_i$) div 7 = (1,3,3,8,1,4,7,7) with remainder 2.

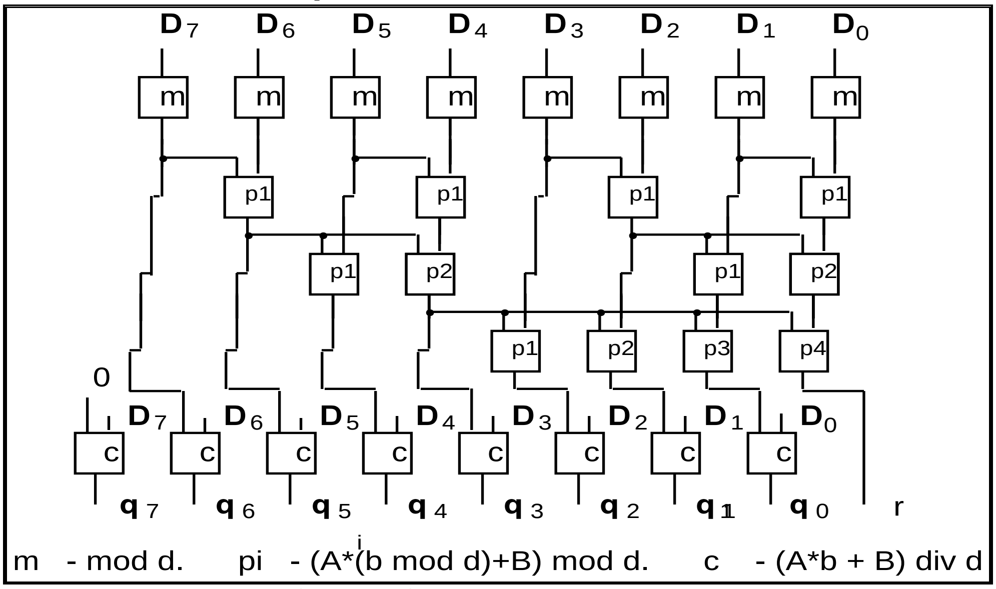


Figure 4: Derivation of quotient from remainders with small constant divisor

5.2 Multiplication by reciprocal

The constant divisor may be factored into two constants as $d = d_1 * d_2$, where d_1 has prime factors that are also factors of the number base b and where d_2 is relatively prime to b. Division by d can be performed by successive division by d_1 and d_2, which need to be treated differently.

Divisor factor of number base

The factor d_1 can be composed into the product of a series of factors c_j such that c_j ² b. Division by d_1 can be performed by successive (or composed) division by c_j. Concentrating on the ith digit:

$D = \ldots \mathbf{D}_{i+1} b^{i+1} + \mathbf{D}_i b^i + \ldots$, so

$D \text{ div } c_j = \ldots (\mathbf{D}_{i+1} \text{ div } c_j + (\mathbf{D}_{i+1} \text{ mod } c_j)/c_j) * b^{i+1} + (\mathbf{D}_i \text{ div } c_j + (\mathbf{D}_i \text{ mod } c_j)/c_j) * b^i + \ldots$

$= \ldots (\mathbf{D}_{i+1} \text{ div } c_j + (\mathbf{D}_{i+1} \text{ mod } c_j)*(b \text{ div } c_j)*b^{-1})*b^{i+1}$

$+(\mathbf{D}_i \text{ div } c_j + (\mathbf{D}_i \text{ mod } c_j)*(b \text{ div } c_j)*b^{-1})*b^i + \ldots$

$= \ldots + ((\mathbf{D}_{i+1} \text{ mod } c_j)*(b \text{ div } c_j) + \mathbf{D}_i \text{ div } c_j)*b^i + \ldots$

The multiplier of b^i in this expression = $(\mathbf{D}_{i+1} \text{ mod } c_j)*(b \text{ div } c_j) + \mathbf{D}_i \text{ div } c_j$. This is non-negative and has maximum value of $((b-1) \text{ mod } c_j)*(b \text{ div } c_j) + (b-1) \text{ div } c_j = (c_j-1)*(b \text{ div } c_j) + (b \text{ div } c_j) - 1 = c_j*(b \text{ div } c_j) - (b \text{ div } c_j) + (b \text{ div } c_j) - 1 = c_j*(b \text{ div } c_j) - 1 = b-1$. Thus the expression above represents the unique representation base b of D div c_j. The value of each digit is found from the local expressions $(\mathbf{D}_{i+1} \text{ mod } c_j)*(b \text{ div } c_j) + \mathbf{D}_i \text{ div } c_j$ which may be calculated in parallel. That is, division by prime factor of the number base is essentially a trivial operation on digit-size numbers. In particular, of course, if $c_j = b$ then the ith digit is $\mathbf{D}_{i+1}$ - division reduces to shifting.

Example:

$\mathbf{D} = (5, 6, 7, 8, 4, 3, 2, 1)$, $d = c_j = 2$ is a factor of the base 10. b div $c_j = 5$

$\mathbf{D}_i \text{ mod } c_j = (1, 0, 1, 0, 0, 1, 0, 1)$, $\mathbf{D}_i \text{ div } c_j = (2, 3, 3, 4, 2, 1, 1, 0)$.

$(\mathbf{D}_{i+1} \text{ mod } c_j)*(b \text{ div } c_j) + \mathbf{D}_i \text{ div } c_j = (0+2, 5+3, 0+3, 5+4, 0+2, 0+1, 5+1, 0+0)$

$= (2, 8, 3, 9, 2, 1, 6, 0)$, remainder 1.

Divisor relatively prime to number base

Division by d_2 is more of a challenge and must, in general, be performed by multiplication by the reciprocal. However, because d_2 is relatively prime to the number base it can be shown that the reciprocal of d_2 is a continued repeating fraction base b. Therefore, multiplication by the reciprocal can be performed by multiplying once by the repeated section, then performing repeated addition. This process is of advantage when the repeated section is tiny (it can be of length up to d_2).

Example:

Divide 240132 by 3 in base 5.

$d_2 = 3$, $d_2^{-1} = 0.13131313.....$

240132*13= 4222321

D = 42223.21 + 422.2321+ 4.222321+ 0.04222321+... = 43210.31....

The additions may be performed in parallel in a tree-like structure but such a circuit is not much more simple than a general multiplier. However, with serial implementation in firmware the technique has certainly been used in practice ([Jacobsohn 1973], [Artzy et. al 1976]) in small computers.

6 Dividend exact multiple of divisor

Having a dividend D ($=D^{(0)}$) that is an exact multiple of the divisor d does not really simplify the division process for standard representations because it is certainly not the case that d divides the other $D^{(i)}$ exactly. However, it can be a help in special cases and with other representations.

In the case of modular representation, if the inverse of X mod M exists and if X = $(....x_i,....)$ then the representation of X^{-1} mod M is $(....x_i^{-1} \bmod m_i....)$. If Y=kX mod M then YX^{-1} mod M is Y div X and is represented by $(....y_i*x_i^{-1}...)$. In other words, exact division works precisely as expected in modular representation. Unfortunately, if Y_kX then YX^{-1} mod M has no obvious relationship to Y div X, nor is there an obvious means of calculating quickly (Y div X)mod M.

A situation where prior knowledge of the remainder can be helpful is in calculating the quotient for small constant divisors that are relatively prime to the number base. Suppose that we know the remainder $R^{(0)}$, then $d*\mathbf{q}_0 = R^{(1)}*b + \mathbf{D}_0 - R^{(0)}$. If d and b are relatively prime there is only one k for which kb + $(\mathbf{D}_0 - R^{(0)})$ is a multiple of d by a factor less than b. We can thus deduce both $\mathbf{q}_0$ and $R^{(1)}$. Having found $R^{(1)}$ we can then deduce $\mathbf{q}_1$ and $R^{(2)}$ from $d*\mathbf{q}_1 = R^{(2)}*b + \mathbf{D}_1 - R^{(1)}$, and so on. Thus it becomes possible to calculate the quotient, and partial remainders, from least significant digit first.

Example:

d = 7, b = 10

Table for solving $d*\mathbf{q}_i = R^{(i+1)}*b + (\mathbf{D}_i - R^{(i)})$:

$(\mathbf{D}_i - R^{(i)})$	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9
$\mathbf{q}_i$	2	5	8	1	4	7	0	3	6	9	2	5	8	1	4	7
$R^{(i+1)}$	2	4	6	1	3	5	0	2	4	6	0	3	5	0	2	4

D = 93670341, $R^{(0)} = 2$

$\mathbf{D}_0 - R^{(0)} = 1-2 = -1$ so $R^{(1)} = 5$, $\mathbf{q}_0 = 7$

$\mathbf{D}_1 - R^{(1)} = 4-5 = -1$ so $R^{(2)} = 5$, $\mathbf{q}_1 = 7$

....

$\mathbf{D}_7 - R^{(7)} = 9-2 = 7$ so $R^{(8)} = 0$, $\mathbf{q}_7 = 1$

The value $R^{(n)}$ is a check on errors in our calculation because $R^{(n)} = 0$ iff the remainder $R^{(0)}$ was correct. If the remainder $R^{(0)}$ is not known, it is possible to calculate from right to left the supposed quotient for each of the d possible remainders and then select as the correct one that for which $R^{(n)} = 0$.

In [Artzy et. al. 1976] the following elegant version of reciprocal multiplication is given.

Suppose d is relatively prime to the number base then, as before, 1/d is a repeating base-b fraction $0.s_1 s_2 ...s_m s_1 s_2 ...s_m s_1 s_2 ...s_m ......$

If $s = s_1 s_2 ...s_m$ then $1/d = s/(b^m-1)$. Now, suppose that D = qd is an exact multiple of d, then $D*s = q*d*s = q*(b^m-1) = q*b^m - q$. So, if $q < b^m$, then q is the base-complement of the last m digits of D*s (because $q*b^m$ has low-order m digits zero).

Example:

$b = 10$, $d = 27$
$1/d = 0.037037037037.... = 037/(10^3-1)$, $s = 037$
If D=1134 then D*s = 41958, so q = 1000-958 = 42

To convert this into a more practical procedure [Artzy et. al 1976] propose first restricting D to be < b^m, in which case q < b^m and it can also be shown that b^m - (b^m-q) ^{2} s only if d divides D exactly, so the error of d not dividing D exactly can be detected. If D^3b^m then the technique is extended to use ss of length 2m, then ssss of length 4m etc., until D is within range, looking at the last 2m, 4m etc digits of D*ss, D*ssss etc. Multiplication by ss, ssss etc. is performed as *s*(b^m+1), *s*(b^m+1)*(b^{2m}+1) etc. (multiplication by b^{km}+1, is, of course, shift and add).

Example:

d = 27, D=1466667, length 7 but s = 037 is length 3 so have to use ssss.
D*037 = 54266679
D*037037 = d*037*1001= 54266679*1001 = 54320945679
D*037037037037 = d*037*1001*1000001 = 54320945679*1000001
= 54320999999945679
So q = 1000000000000 - 999999945679 = 54321.
54321< 037037037037, so the division was indeed exact.

A practical use of division where it is known that d divides D exactly is where D is a, for example, byte offset previously constructed by multiplication of an offset q by the element size (in bytes) d.

7 Divisor related to the number base

We have encountered already some simplifying relationships between the divisor and number base. These included cases where the divisor divides the base exactly and where the divisor and the base are mutually prime. Another relationship of interest is when the divisor is close to the base in value.

7.1 Divisor close to number base in value

If the divisor is very close to the base, then it is possible to find the first digit of the quotient very quickly because it is the first digit of the dividend. This process may be continued and used to calculate the next quotient digit at each stage of the division process. The example below has the divisor close to the base but works with fixed point fractions. The same procedure may be applied to integer division where the divisor is close to some power of the base i.e. the same procedure with the point shifted to the right.

Example:

d = 9.934 , D = $R^{(3)}$ =	**5**678
$\mathbf{q}_2$ = 5, $R^{(2)}$ = 5678 - 993.4*5 =	**7**11.0
$\mathbf{q}_1$ = 7, $R^{(1)}$ = 711.0 - 99.34*7=	**1**5.62
$\mathbf{q}_0$ = 1, $R^{(0)}$ = 15.62 - 9.934*1=	5.686

5678 = 571*9.934 + 5.686

This process is an important one that forms the basis of some of the fastest practical division algorithms [Ercegovac et. al. 1983] .

A case of particular interest is when the divisor is the constant b-1 or b+1. In fact, when the representation is binary many small integer divisors are in this form or are related to it. A binary number may be regarded as being base 4 (each digit of 2 bits), base 8 (each digit of 3 bits) etc. Division by 3 is by 4-1, by 5 4+1, by 7 8-1, by 9 8+1, by 17 16+1 etc

7.2 Division by Base-1

In this case remaindering is very simple. Because b = d+1, b mod d = 1 and so b^k mod d =1. Hence

$$R^{(i)} = D^{(i)} \bmod d = (\mathbf{D}_{n-1}\, b^{n-1-i} + \mathbf{D}_{n-2}\, b^{n-2-i} + \ldots\ldots + \mathbf{D}_i\, b^0) \bmod d$$
$$= (\mathbf{D}_{n-1} + \mathbf{D}_{n-2} + \ldots\ldots + \mathbf{D}_i) \bmod d.$$

The reduction of the sum of digits mod d can be performed serially with the recurrence:

$$R^{(i)} = (R^{(i+1)} + \mathbf{D}_i) \bmod d$$

or can be calculated with parallel tree circuits. If solely $R^{(0)}$ is required then one can use a tree circuits as in [Fig. 2]. If all $R^{(i)}$ are required a circuit as in the top part of [Fig. 5] is appropriate.

The calculation of the remainder mod b-1 is used in checking arithmetic in the classic "casting out nines" procedure. For example, to check the full-sized multiplication A*B=C the relationship ((A mod d) * (B mod d)) mod d = C mod d is tested, where the multiplication is much simpler. In the manual approach the digits are not summed modularly digit by digit but rather they are summed using normal arithmetic and the process reapplied repeatedly until one digit remains:

Example:

D = 93608719, b = 10, d = 9
D mod 9 = (9+3+6+0+8+7+1+9) mod 9 = 43 mod 9 = (4+3) mod 9 = 7

The same approach is used in computers for error checking. Usually, a binary number is regarded as being in base 4 and the remainder found mod 3. The digits 0, 1, 2 are kept in decoded form 100, 010, 001, to simplify the nodes in the tree circuit.

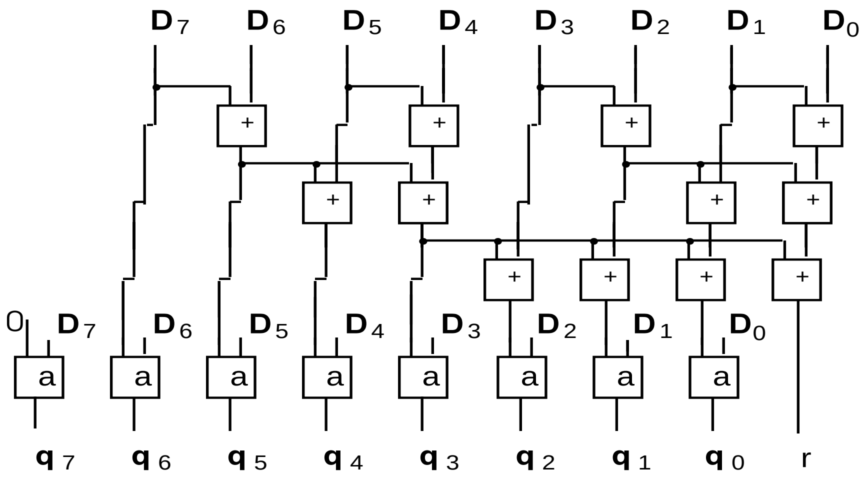
Proceeding further, with the remainders known we can find the quotient as in section 1.2 from $R^{(i+1)}*b+\mathbf{D}_i = d*\mathbf{q}_i + R^{(i)}$. We have:

$$d*\mathbf{q}_i = R^{(i+1)}*b + \mathbf{D}_i - R^{(i)} = R^{(i+1)}*d + R^{(i+1)} + \mathbf{D}_i - R^{(i)}, \text{ so}$$
$$\mathbf{q}_i = R^{(i+1)} + (\mathbf{D}_i + R^{(i+1)} - R^{(i)}) \text{ div } d$$
$$= R^{(i+1)} + (R^{(i+1)} + \mathbf{D}_i - (R^{(i+1)} + \mathbf{D}_i) \bmod d) \text{ div } d$$
$$= R^{(i+1)} + (R^{(i+1)} + \mathbf{D}_i) \text{ div } d$$

To summarise:

$$R^{(i)} = (R^{(i+1)} + \mathbf{D}_i) \bmod d$$
$$\mathbf{q}_i = R^{(i+1)} + (R^{(i+1)} + \mathbf{D}_i) \text{ div } d$$

In words, the quotient digit i is the remainder adjusted by 1 if $R^{(i+1)} + \mathbf{D}_i$ ³ d.This correction may be applied in parallel as a single step as in [Fig. 5] for n=8.



+ - sum mod d a - A + (A+B) div d

Figure 5: Quotient from remainders for d = b-1

In the circuit of figure 5 there is some duplication of effort in that the calculation of the final adjustment could well be associated with the previous stage.This idea is used in the following serial algorithm where the quotient is calculated as we proceed with determining the partial remainders. This algorithm is particularly nice in that the division is performed entirely by digit addition and comparison ³ d (in the algorithm all arithmetic is standard).

{find R := D mod b-1 and **q** the digits of D div b-1}
R := 0
Repeat for i from n-1 down to 0
T := R + $\mathbf{D}_i$
{$\mathbf{q}_i$:= R + T div (b-1), R := T mod (b-1)}
if T ³ b-1 then T := T+1
$\mathbf{q}_i$:= R + T div b {the b^1 digit of T}
R := T mod b {the b^0 digit of T}

Example: Division of a base 10 number by 9.

	9	3	6	0	8	7	1	9	dividend
0	0	3	0	0	8	6	7	7	remainders
	1	0	1	0	0	1	0	1	adjustments
	1	0	4	0	0	9	6	8	quotient

Typical step:
T := 8 + 7 (=15)
if 15 ³ 9 then T := 15+1 = 16
$\mathbf{q}_i$:= 8 + 1
R := 6

Note that it is also possible to calculate the remainders from right to left. From $R^{(i)}$ = ($R^{(i+1)}$ + $\mathbf{D}_i$) mod d, we find $R^{(i+1)}$ = ($R^{(i)}$ - $\mathbf{D}_i$) mod d. Continuing this expansion we get $R^{(i+1)} = (R^{(0)} - (\mathbf{D}_i + \mathbf{D}_{i-1} \cdots + \mathbf{D}_0)) \bmod d$. Finally, $R^{(n)} = (R^{(0)} - (\mathbf{D}_{n-1} + \mathbf{D}_{n-2} \cdots + \mathbf{D}_0)) \bmod d$.

If we knew $R^{(0)}$ correctly then we would have $R^{(n)} = 0$. However, if we did not know $R^{(n)}$ but assumed it zero then we will have $R^{(n)} = (-R^{(0)}) \bmod d$. The remainders that we have found will be incorrect but they can be restored by now adding $R^{(n)} \bmod d$. This approach is inherent in the circuit of [Duke 1972] , described below.

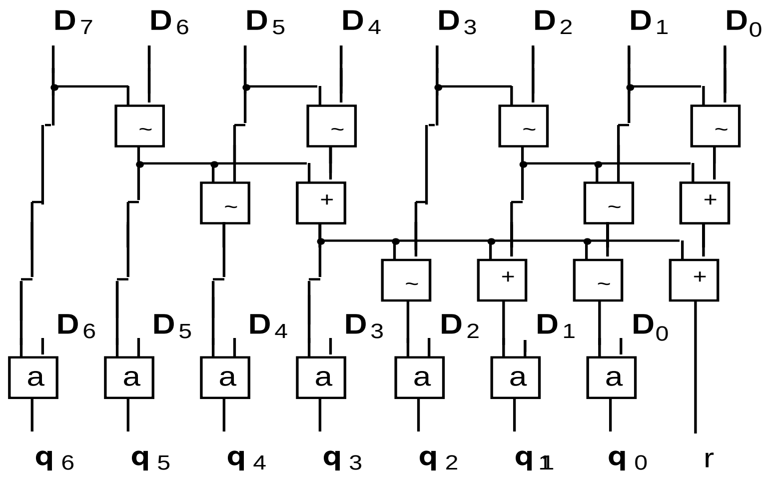
7.3 Division by Base+1

The above reasoning can be revisited with d = b+1. Noting that (A-B) mod d is the non-negative integer C such that (B+C) mod d = A mod d, in this case we find:

$R^{(i)} = (\mathbf{D}_i - R^{(i+1)}) \bmod d$

$\mathbf{q}_i = R^{(i+1)} + (\mathbf{D}_i - R^{(i+1)}) \operatorname{div} d$

Without going into details, [Fig. 6] is an example circuit.



+ - (A+B) mod d ~ - (B-A) mod d a - A + (B-A) div d

Figure 6: Quotient from remainders for d = b+1

And the serial version is (in standard arithmetic):

{find R := D mod b+1 and **q** the digits of D div b+1}
R := 0
Repeat for i from n-1 down to 0
T := $\mathbf{D}_i$ - R
{$\mathbf{q}_i$:= R + T div (b+1), R := T mod (b+1)}
if T ³ 0 then $\mathbf{q}_i$:= R, R := T
else $\mathbf{q}_i$:= R-1, R := T + b+1

Example: Division of a base 10 number by 11:

```
  9 3 6 0 8 7 1 9     dividend
0 9 5 1 10 9 9 3 6    remainders
  0 1 0 1 1 1 1 0     adjustments
  0 8 5 0 9 8 8 3     quotient
```

Typical step:

T := 7 - 9 (= -2)

if -2 ³ 0 else $\mathbf{q}_i$:= 9 - 1 (=8), R := -2 + 11 (=9)

The above two serial algorithms were described in [Doran 1987] though were presumably known to mental calculators previously. Surprisingly, division and remaindering by b+1 has had at least one practical[2] application in the proposed Burroughs Scientific Processor of the 1970s which had 17 memory banks (b = 16).

7.4 Circuits with feedback

[Duke 1972] describes the circuit illustrated in [Fig. 7].

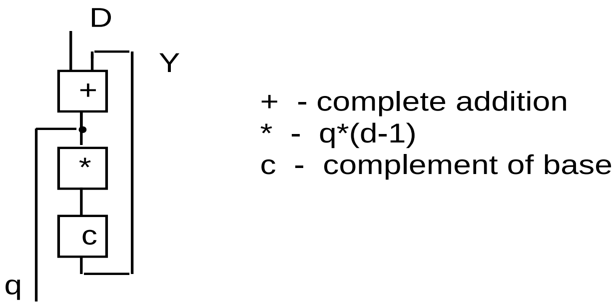


Figure 7: Duke’s feedback division circuit

This is unusual in that it is a logic circuit with feedback. If it ever produces a stable output then we must have:

$$q = D+Y,\ Y = -q*(d-1) \text{ and so } q = D-q*(d-1) \text{ and } q*d=D$$

If d divides D exactly then q = D div d.

The question is, when is the output stable? This answer is clearly technology and implementation dependent. In the case of binary representation, the circuit will certainly work when "*(d-1)" introduces a genuine shift. This is when $d-1 = 2^k$, so $d = 2^k+1$. In this case, although at the overall level the circuit has feedback, at the bit level it can be shown that it does not, so is definitely combinational. It is, in fact, a circuit implementation of the algorithm described above for division by b+1 when the remainder is zero but from the right (in a manner analagous to that explained for division by b-1)

The above circuit can be adapted for $d = 2^k + 1$ in the obvious manner. If d does not divide D exactly, then [Duke 1972] showed that the high order carry is non-zero and

---

[2] One could note in passing that in New Zealand the VAT (called GST) had been set at 10% initially then changed to 12.5%. To find the tax included in the purchase price the first rate needed division by 11 and the second by 9 - so these algorithms would have been useful, if we did not have calculators!

provides the correction factor to be added to each digit (also as described above for d = b-1).

[Fenwick 1972] mentions that Boothroyd had proposed a similar circuit with feedback working from the most significant end. In this case, the shift is to the right. Unfortunately, because carry is to the left, this involves real feedback and cannot be guaranteed to stabilise.

7.5 Application to Binary to Decimal Conversion

We can at last see the practical use of division by a small constant for base conversion. The basic idea is that if we wish to convert a number D from a base b to a base d, then, because D = q*d+r, r is the zeroth digit of the representation base d. The process may then be applied to q to get the first digit, etc. If the arithmetic for the division is performed in base b, then the sequence of remainder digits are the representation in base d, with each digit represented in base b.

Example:

To convert N=110001 (49) to BCD:
110001/1010(ten) = 0100, remainder 1001 (nine)
0100/1010(ten) = 0000 remainder 0100 (four)
So, 110001 (base 2) = 0100,1001 (BCD) = 49 (base 10)

The most common case is binary-to-decimal conversion which requires division by 10. This may be performed by dividing the binary number by 5 then by 2. In both divisions the remainders are also found. If the first division is $D = 5*Q+R_5$ and the second is $Q = 2*q + R_2$, the combined effect is $D = 5*(2*q+ R_2) + R_5 = 10*q + (5*R_2 + R_5)$. Thus, as R_2 is 0 or 1, the remainder mod 5 is increased by 5 if the remainder mod 2 is 1, to give the remainder mod 10 which is the next decimal digit. Because $5 = 2^2+1$, we can regard the binary number as being base b=4 and use our circuits for division by b+1, and the division by 2 is, of course, a shift to the right.

If conversion is required to be performed for one particular number then a tree division by b+1 circuit would be appropriate. Another technique, more appropriate for VLSI is to use a *cellular* circuit, where the approach is to use a network of idential components or cells that are connected in a two dimensional grid. Cellular binary-to-decimal circuits are explored in detail in [Schreiber and Stefanelli 1978]. To see an example, we could base a cellular circuit on our repetitive algorithm for b+1 division described above. The circuit consists of multiple levels, each performing a division by 10 and production of the next decimal digit encoded in BCD. The cell corresponds to the inner loop of the algorithm and performs a mod 5 subtraction of the incoming partial remainder (represented in 3 bits as being in range 0 to 4) with the next 2-bit digit of the binary number, followed by correction of the quotient digit based on the subtraction being negative (if it were not modular). Without fully developing the logic, the cell would be as in [Fig. 8].

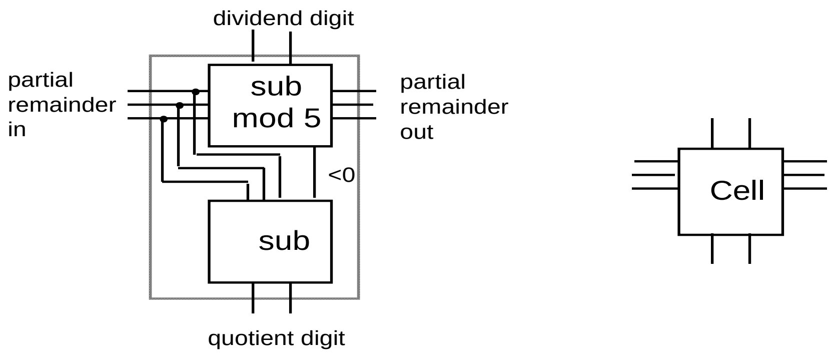


Figure 8: Cell for Binary to Decimal Conversion

The cells are then combined into the grid, with division by 2 being performed by directing the wires to perform a shift. An array of special circuits on the right is needed to conditionally increment the digits by 5. Each level of division needs less circuitry. [Fig. 9] shows the example of 8-bit conversion (maximum value 255).

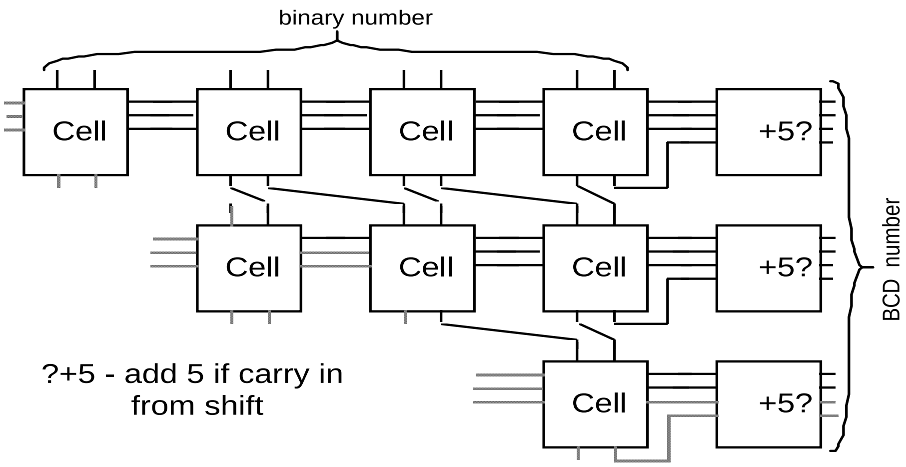


Figure 9: 8-bit cellular binary to decimal converter

The circuit above is somewhat simpler than that presented in [Schreiber and Stefanelli 1978].

8 Conclusion

There have certainly been many interesting procedures proposed for special cases of division. We have seen that these are mainly variations on two themes, division by reciprocal multiplication, or division derived from remaindering.

Of course, for every specific constant divisor there are special tricks that can be brought to bear. These have been developed over the years by mental calculators, see [Aiken 1937] , [Menninger 1964], [Smith 1983], [Yang 1274] . This is a fascinating but endless pursuit that we will leave it to the reader to take further.

Acknowledgments

This work was done while the author was a visitor with the Laboratoire d'Informatique du Parallélisme at the École Normale Supérieure at Lyon, France. Thanks is due to Jean-Michel Muller of LIP ENS and to Elena Calude and Radu Nicolescu of The University of Auckland for helpful suggestions.

References

[Aiken 1937] Aiken, A. C.: "Trial and error and approximation in arithmetic"; The Mathematical Gazette (1937), p. 117.

[Alt and Blum 1983] Alt, H., Blum, N.: "On the Boolean circuit depth of division and related functions"; Dept. of Computer Science, Pennsylvania State University (1983).

[Artzy et. al. 1976] Artzy, E., Hinds, J. A., & Saal, H. J. : "A fast division technique for constant divisors"; CACM, 19, 3 (1976), 98 - 101.

[Beame, Cooke and Hoover 1986] Beame, P. W., Cook, S. A. & Hoover, H.J.:"Log depth circuits for division and related problems"; SIAM Journal on Computing, 15 (1986) 994-1003.

[Cocke et. al. 1970] Cocke, J., Freiman C. V., & Homan M. E.: "High speed division system"; US Patent # 3,527,930 (1970).

[Daumas et. al. 1994] Daumas, M., Mazenc, C., Merrheim, X, and Muller, J-M.: "Fast and Accurate Range reduction for thr computation of elementary functions"; 14th IMACS World Congree on Computational and Applied Mathemaatics, Atalanta, Georgia (1994).

[Doran 1987] Doran, R. W.: "Parallel division circuits for small divisors"; Tech Report No. 38. Department of Computer Science, University of Auckland (1987).

[Duke 1972] Duke, K. A.: "Division by small integers"; IBM Technical Disclosure Bulletin, 14, 9 (1972), 3736-2738.

[Ercegovac et. al. 1993] Ercegovac, M.D., Lang, T., Montuschi, P.: "Very high radix division with selection by rounding and prescaling"; Proceedings of the 11th Symposium on Computer Arithmetic, Windsor, Ontario (1993) 112 - 119.

[Fenwick 1972] Fenwick, P.McA.: "A binary representation for decimal numbers"; The Australian Computer Journal, 4, 4 (1972), 146 - 149.

[Iliffe 1982] Iliffe, J. K.: "Advanced Computer Design"; Prentice Hall. London (1982).

[Jacobsohn 1973] Jacobsohn, D. H.: "A combinatoric algorithm for fixed-integer divisors"; IEEE Transactions on Computers (1973), 608 - 610.

[Menninger 1964] Menninger, K.: "Calculator's Cunning - The Art of Quick Reckoning"; G. Bell and Sons Ltd., London (1964).

[Schreiber and Stefanelli 1978] Schreiber F. A., Stefanelli, R.: "Two methods for fast integer binary-BCD conversion"; Proceedings of 4th Symposium on Computer Arithmetic, Santa Monica, California (1978), 200-207.

[Smith 1983] Smith, S. B.: "The Great Mental Calculators - The Psychology, Methods, and Lives of Calculating Prodigies Past and Present"; Columbia University Press, New York (1983).

[Szabo and Tanaka 1967] Szabo, N. S. & Tanaka, R. I.: "Residue Arithmetic & its Applications to Computer Technology"; McGraw Hill, New York (1967).

[Yang 1274] Yang Hui: "Ch'eng Chu' T'ung Pien Suan Pao (Precious Reckoner for Variations of Multiplication and Division)"; Reprinted, translated with commentary, in Lam Lay Yong, A Critcal Study of the Yang Hui Suan Fa, Singapore University Press (1977).