Division
Ivor Page[1]

12.1 Division

Division is difficult for humans and computers. In module on multi-precision arithmetic, the division algorithm is by far the most complex. The pencil and paper method for decimal values requires us to guess the next digit of the quotient at each iteration. The divisor is then multiplied by that quotient digit and the result is subtracted from the partial remainder. The guess must be adjusted if it is too high or too low and the multiply and subtract steps repeated.

In a binary computer the dividend is a double-length integer and the divisor is a single-length integer. The result is a quotient and a remainder, both single-length integers.

Here is the notation:

$$\begin{aligned}
\textit{Dividend } z &= z_{2k-1}z_{2k-2}\cdots z_1z_0\\
\textit{Divisor } d &= d_{k-1}d_{k-2}\cdots d_1d_0\\
\textit{Quotient } q &= q_{k-1}q_{k-2}\cdots q_1q_0\\
\textit{Remainder } s &= s_{k-1}s_{k-2}\cdots s_1s_0\\
s &= z - d\times q\\
sign(s) &= sign(z)
\end{aligned}$$

The last equation ensures that division is the inverse of multiplication (plus the remainder).

To avoid overflow, $z < 2^k d$ must be true. Therefore the high-order k bits of z must be strictly less than d.

The algorithm repeatedly applies the recurrence relation:

$$s^{(j)} = 2s^{(j-1)} - q_{-j}d$$

where $s^{(0)} = z$. The notation $s^{(k)}$ means $2^k s$.

$$q_i = \begin{cases} 0 & : \quad s^{(i)} < d \\ 1 & : \quad s^{(i)} \geq d \end{cases}$$

Note: I have used the notation from the text in this section, but you might want to consider writing $2^k s_k$ in place of $s^{(k)}$. The latter does not make it clear that the k^{th} iteration of s is implied. The term 2^k corresponds to the left shifts of the partial remainder on each iteration.

[1]University of Texas at Dallas

Here is a decimal example:

```
               523 ) 9743854
                  -  523       d*10000
                     -------
                     4513854   = partial remainder
                  -  4184      d*8000
                     -------
                      329854
                   -  3138     d*600
                      ------
                       16054
                    -  1569    d*30
                       -----
                         364   remainder, quotient = 18630
```

After two iterations, $z = (d \times q_i) + s_i = 523 \times 18000 + 329854$.

In binary the next digit of the quotient is either 0 or 1:

```
        10011 ) 1010110111011   = 5563/19
              - 10011           d*2^8, q8=1
                -------------
                   1010111011
                 - 10011        d*2^5, q5=1
                 ------------
                      1011011
                    - 10011     d*2^2, q2=1
                   ----------
                         1111   remainder = 1111,
                                quotient = 100100100
```

The example shows only those iterations in which the quotient is changed.

In this section we will only consider positive operands.

The notation used in the problems above is suited to pencil and paper calculations. Now we build up to the hardware shift-subtract time-iterative divider.

Fractional division and integer division are achieved with exactly the same steps:

Integer Division z = 117, d = 10			Fractional Division z = 0.45703125, d = 0.625		
z	0 1 1 1 0 1 0 1		z_{frac}	.0 1 1 1 0 1 0 1	
$d2^4$	1 0 1 0		d_{frac}	.1 0 1 0	
$s^{(0)}$	0 1 1 1 0 1 0 1		$s^{(0)}$	.0 1 1 1 0 1 0 1	
$2s^{(0)}$	0 1 1 1 0 1 0 1		$2s^{(0)}$	0.1 1 1 0 1 0 1	
$-q_3d2^4$	1 0 1 0	$q_3 = 1$	$-q_{-1}d$	.1 0 1 0	$q_{-1} = 1$
$s^{(1)}$	0 1 0 0 1 0 1		$s^{(1)}$	.0 1 0 0 1 0 1 0	
$2s^{(1)}$	0 1 0 0 1 0 1		$2^s(1)$	0.1 0 0 1 0 1	
$-q_2d2^4$	0 0 0 0	$q_2 = 0$	$-q_{-2}d$	.0 0 0 0	$q_{-2} = 0$
$s^{(2)}$	1 0 0 1 0 1		$s^{(2)}$	0.1 0 0 1 0 1	
$2s^{(2)}$	1 0 0 1 0 1		$2s^{(2)}$	1.0 0 1 0 1	
$-q_1d2^4$	0 0 0 0	$q_1 = 1$	$-q_{-3}d$	.1 0 1 0	$q_{-3} = 1$
$s^{(3)}$	1 0 0 0 1		$s^{(3)}$	.1 0 0 0 1	
$2s^{(3)}$	1 0 0 0 1		$2s^{(3)}$	1.0 0 0 1	
$-q_0d2^4$	1 0 1 0	$q_0 = 1$	$-q_{-4}d$	.1 0 1 0	$q_{-4} = 1$
$s^{(4)}$	0 1 1 1		$s^{(4)}$	.0 1 1 1	
s	0 1 1 1		s_{frac}	.0 0 0 0 0 1 1 1	
q	1 0 1 1		q_{frac}	.1 0 1 1	
$q = 11, r = 7$			$q = 0.6875, r = 0.02734375$		

The example shows the basic steps of subtraction and left shift. The value of d is scaled by 2^k (left shifted k places) at the beginning. This happens naturally in the registers used during division.

A comparison is implied between $2s^{(i)}$ and $q_{(k-i-1)}d2^k$, $i = 0, 1, \cdots k-1$. In practice, the hardware always subtracts or adds. This leads to two simple time-iterative division algorithms that cause a subtraction or addition on every cycle.

Restoring Division keeps the partial remainder $s^{(i)}$ positive and performs up to k cycles in which a subtraction and, sometimes, an addition takes place.

In **Non-Restoring Division** each cycle requires either an add or a subtract.

12.2 Restoring Division For Positive Operands

1. Initially s is the dividend, d is the divisor, the quotient $q = 0$, and $i = 0$
2. left shift d by k places giving $d2^k$.
3. left shift s one place
4. subtract $d2^k$ from s
5. if the result is non-negative set the quotient bit, $q_{k-i-1} = 1$
6. else add $d2^k$ to s undoing the subtraction step
7. increment i by 1
8. if $i < k$ got to step 3

At the end of the algorithm, s is the remainder and q is the quotient.

12.3 Non-Restoring Division For Operands > 0

1. Initially s is the dividend, d is the divisor, the quotient $q = 0$, and $i = 0$
2. left shift d by k places giving $d2^k$
3. left shift s one place
4. if $s > 0$ subtract $d2^k$ from s and set $q_{k-i-1} = 1$
5. else add $d2^k$ to s
6. increment i by 1
7. if $i < k$ got to step 3
8. if $s < 0$ add d to s.

At the end of the algorithm, s is the remainder and q is the quotient.

We will refine these simple descriptions of the algorithms for the hardware we will use.

12.4 Hardware Organization

Figure 1 shows the hardware setup. It supports restoring or non-restoring division.

Initially the dividend z is loaded into the double register u, v.

If $u > d$, then the quotient is larger than k bits and an overflow results.

Each cycle comprises a left shift of u, v followed by a subtraction (or addition) of the divisor d from u.

Subtraction is achieved by is setting the carry-in of the adder to 1 and presenting the 1's complement of the divisor to the adder.

If $c_{out} = 1$, the result is positive and a 1 is inserted into the lsb of the v register. Otherwise a 0 is inserted. After each addition or subtraction c is set equal to the carry-out of the adder.

After k cycles, if $c = 0$ an extra add occurs to restore the remainder. The quotient is in v and the remainder is in u.

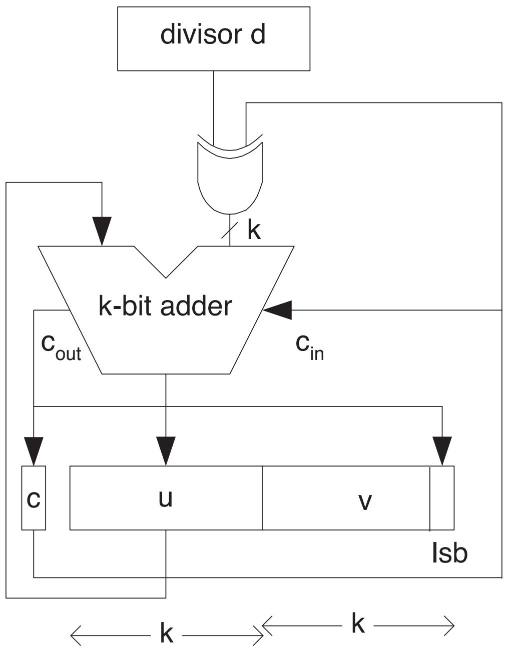

Figure 1: Shift/Add Divider

12.5 Example of Restoring Division with above hardware

Integer Division

z = 117, d = 10

z	0 1 1 1	0 1 0 1	No overflow since (0111)¡(1010)
$d2^4$	0 1 0 1 0		
$-d2^4$	1 0 1 1 0		
$s^{(0)}$	0 0 1 1 1	0 1 0 1	
$2s^{(0)}$	0 1 1 1 0	1 0 1	
$+(-d2^4)$	1 0 1 1 0		Subtract
$s^{(1)}$	0 0 1 0 0	1 0 1	Positive so set $q_3 = 1$
$2s^{(1)}$	0 1 0 0 1	0 1	
$+(-d2^4)$	1 0 1 1 0		Subtract
$s^{(2)}$	1 1 1 1 1	0 1	Negative so set $q_2 = 0$
$s^{(2)} = 2s^{(1)}$	0 1 0 0 1	0 1	and restore
$2s^{(2)}$	1 0 0 1 0	1	
$+(-d2^4)$	1 0 1 1 0		Subtract
$s^{(3)}$	0 1 0 0 0	1	Positive so set $q_1 = 1$
$2s^{(3)}$	1 0 0 0 1		
$+(-d2^4)$	1 0 1 1 0		Subtract
$s^{(4)}$	0 0 1 1 1		Positive so set $q_0 = 1$
s		0 1 1 1	
q		1 0 1 1	

$q = 11$, $r = 7$

12.6 Example of Non-Restoring Division with above hardware

Integer Division

z = 117, d = 10

z	0 1 1 1	0 1 0 1	No overflow since (0111)¡(1010)
$d2^4$	0 1 0 1 0		
$-d2^4$	1 0 1 1 0		
$s^{(0)}$	0 0 1 1 1	0 1 0 1	Positive so subtract
$2s^{(0)}$	0 1 1 1 0	1 0 1	
$+(-d2^4)$	1 0 1 1 0		
$s^{(1)}$	0 0 1 0 0	1 0 1	Positive so set $q_3 = 1$
$2s^{(1)}$	0 1 0 0 1	0 1	and subtract
$+(-d2^4)$	1 0 1 1 0		
$s^{(2)}$	1 1 1 1 1	0 1	Negative so set $q_2 = 0$
$2s^{(2)}$	1 1 1 1 0	1	and add
$+d2^4$	0 1 0 1 0		
$s^{(3)}$	0 1 0 0 0	1	Positive so set $q_1 = 1$
$2s^{(3)}$	1 0 0 0 1		and subtract
$+(-d2^4)$	1 0 1 1 0		
$s^{(4)}$	0 0 1 1 1		Positive so set $q_0 = 1$
s		0 1 1 1	
q		1 0 1 1	

$q = 11$, $r = 7$

The examples show the need for an extra bit to the left of the u, v register pair.

In the non-restoring example the final remainder was positive, which was correct since its sign must match the sign of the dividend, z. If we had ended up with a negative remainder, an add cycle would have been needed to correct the result and the quotient would have been reduced by 1.

12.7 Signed Numbers

The non-restoring algorithm can easily be modified to deal with signed numbers. We wish to produce a remainder that has the same sign as the dividend, sign(s(k)) = sign(z).

The rule for quotient digit selection is as follows:

If sign(s) = sign(d) then q(k-i-1) = 1 else q(k-i-1) = -1

If the final step leaves the signs of s and z different a corrective step must be applied.

Here is an example:

			Integer Division
			z = 33, d = -7
z	0 0 1 0	0 0 0 1	No overflow since (0111) < (1010)
$d2^4$	1 1 0 0 1		
$-d2^4$	0 0 1 1 1		
$s^{(0)}$	0 0 0 1 0	0 0 0 1	
$2s^{(0)}$	0 0 1 0 0	0 0 1	$sign(s^{(0)}) \neq sign(d)$
$+d2^4$	1 1 0 0 1		so set $q_3 = -1$ and add
$s^{(1)}$	1 1 1 0 1	0 0 1	
$2s^{(1)}$	1 1 0 1 0	0 1	$sign(s^{(1)}) = sign(d)$
$+(-d2^4)$	1 0 1 1 0		so set $q_2 = 1$ and subtract
$s^{(2)}$	0 0 0 0 1	0 1	
$2s^(2)$	0 0 0 1 0	1	$sign(s^{(2)}) \neq sign(d)$
$+d2^4$	1 1 0 0 1		so set $q_1 = -1$ and add
$s^{(3)}$	1 1 0 1 1	1	
$2s^{(3)}$	1 0 1 1 1		$sign(s^{(3)}) = sign(d)$
$+(-d2^4)$	0 0 1 1 1		so set $q_0 = 1$ and subtract
$s^{(4)}$	1 1 1 1 0		$sign(s^{(4)}) \neq sign(z)$
$+(-d2^4)$	0 0 1 1 1		Corrective subtraction
s	0 0 1 0 1		Remainder
q		T 1 T 1	Uncorrected BSD quotient = -5
			where T represents -1
p		0 1 0 1	Replace -1s in q by 0s
		/ / / /	Complement sign, left shift 1 place, set LSB to 1
p'		1 1 0 1 1	Add 1 to correct
$q = p' + 1$		1 1 0 0	Two's complement value of q in k bits
			q = -4, r = 5

The BSD quotient was converted to binary and incremented (to -4).

The conversion of q to binary can be achieved on the fly as the digits are computed.

OR

The following algorithm can be applied to the BSD value of q:

- Replace all the -1 digits by 0s to get the k bit number $p = p_{k-1}p_{k-2}...p_0$ with $p_i \in \{0, 1\}$
- Complement p_{k-1} and then left shift p by one bit, inserting 1 into the LSB to get the 2's complement quotient $q = (\overline{p_{k-1}}p_{k-2}...p_0 1)_{two's\ complement}$

The sense of the corrective step (to add or subtract $d2^k$) is to make the sign of $s^{(k)}$ equal the sign of z.

12.8 SRT Division for fractions

The SRT scheme, named after its inventors, Sweeney, Robertson, and Tocher, skips over sequences of 1s or 0s in the partial remainder register pair u, v.

If, after an addition or subtraction, $s^{(i)}$ is positive and its most significant bit is zero, there is little point in doing another subtraction cycle. Therefore, the u, v register can be left shifted one place.

By similar reasoning, if, after an addition or subtraction, $s^{(i)}$ is negative, and its most significant bit is one, the u, v register can be left shifted one place without needing an add or a subtract in that cycle.

The SRT algorithm operates correctly with signed operands. During each iteration, an addition or a subtraction takes place such that the partial remainder is always reduced towards zero. For example, if the partial remainder is negative and the divisor is also negative, then a subtraction takes place. In general, subtraction is used in those cycles where the signs of the partial remainder and the divisor are the same. Addition is used when their signs differ. After termination, a correction may be necessary to deliver a remainder that has the same sign as the dividend.

12.8.1 Example

The SRT division algorithm keeps the partial remainder in the range $[-1/2, 1/2)$.

$$1.101011101110110 0_2 \div 0.10101100_2$$

In this version the quotient digits are in {-1,1}.

Integer Division

z = 1.1010111011101100, d = 0.10101100

z	1.10101110	11101100	
$d2^8$	0.10101100		
$-d2^8$	1.01010100		
$s^{(0)}$	1.10101110	11101100	
$2s^{(0)}$	1.01011101	1101100	$2s^{(0)} < 0.5$, add, $q_{-1} = -1$
$+d2^8$	0.10101100		
$s^{(1)}$	0.00001001	1101100	
$2s^{(1)}$	0.00010011	101100	$2s^{(1)}$ in $[-0.5, 0.5)$, skip, $q_{-2} = 0$
$s^{(2)}$	0.00010011	101100	
$2s^{(2)}$	0.00100111	01100	$2s^{(2)}$ in $[-0.5, 0.5)$, skip, $q_{-3} = 0$
$s^{(3)}$	0.00100111	01100	
$2s^{(3)}$	0.01001110	1100	$2s^{(3)}$ in $[-0.5, 0.5)$, skip, $q_{-4} = 0$
$s^{(4)}$	0.01001110	1100	
$2s^{(4)}$	0.10011101	100	$2s^{(4)} > 0.5$, subt, $q_{-5} = 1$
$+(-d2^8)$	1.01010100		
$s^{(5)}$	1.11110001	100	
$2s^{(5)}$	1.11100011	00	$2s^{(5)}$ in $[-0.5, 0.5)$, skip, $q_{-6} = 0$
$s^{(6)}$	1.11100011	00	
$2s^{(6)}$	1.11000110	0	$2s^{(6)}$ in $[-0.5, 0.5)$, skip, $q_{-7} = 0$
$s^{(7)}$	1.11000110	0	
$2s^{(7)}$	1.10001100		$2s^{(7)}$ in $[-0.5, 0.5)$, skip, $q_{-8} = 0$
$s^{(8)}$	1.10001100		

quotient = 0.T0001000 − > 1.10001000 in 2's complement

remainder = 1.10001100

Expressed as an integer calculation, the above represents $-20756 \div 172 = -120$, remainder -116

12.8.2 Using a Carry-Save Adder

Unfortunately if the partial remainder is kept in carry-save form where the partial remainder u register becomes two registers, u_{sum} and u_{carry}, the sign of the partial remainder is not immediately available. In the next module we shall see how to estimate the sign and value of the partial remainder and discover what to do when our estimate is wrong.