EE 3755

Computer Arithmetic

Handout # 3

EE [redacted] 3755

Fixed Point Division

Introduction

Consider a $2n$-bit number $A = a_{2n-1} a_{2n-2} \cdots a_1 a_0$ called dividend and an n-bit number $B = b_{n-1} b_{n-2} \cdots b_1 b_0$ called divisor. The division of A by B will result in two numbers: an n-bit number $Q = q_{n-1} q_{n-2} \cdots q_1 q_0$ called quotient and an n-bit number $R = r_{n-1} r_{n-2} \cdots r_1 r_0$ called remainder. The dividend A, divisor B, quotient Q and remainder R satisfy the following

$$\boxed{A = B \times Q + R} \quad (1)$$

where

$$\boxed{R < B} \quad (2)$$

If the numbers are signed, then the remainder R has the same sign as the dividend A. The quotient Q is positive if the dividend A and divisor B are of the same sign (both positive or both negative) while the quotient is negative if the dividend and divisor are of different signs (one positive and one negative).

Division Overflow

Consider the problem of dividing a $2n$-bit dividend A by an n-bit divisor B in order to obtain an n-bit quotient Q and an n-bit remainder R.

A division overflow occurs if the resulting quotient Q needs more then n bits for its representation.

Consider a 2n-bit dividend $A = a_{2n-1}a_{2n-2}\cdots a_n a_{n-1}\cdots a_0$ and consider its [redacted] left-most n-bit part A_1 and its right-most n-bit part A_0, where $A_1 = a_{2n-1}a_{2n-2}\cdots a_n$ and $A_0 = a_{n-1}\cdots a_1 a_0$. Obviously the value of A is $A = A_1 \times 2^n + A_0$. Consider now the division of A by an n-bit divisor B. Then the following two statements hold true:

Statement 1: "If $A_1 \geqslant B$, then a division overflow occurs".

Statement 2: "If $A_1 < B$, then no division overflow occurs".

Statements 1 and 2 dictate that: "The necessary and sufficient condition for a division overflow to occur is that $A_1 \geqslant B$".

Proof of Statement 1:

Consider the division of a 2n-bit dividend $A = a_{2n-1}\cdots a_0 = A_1 A_0$ by an n-bit divisor $B = b_{n-1}\cdots b_1 b_0$ where $A_1 = a_{2n-1}a_{2n-2}\cdots a_n$ and $A_0 = a_{n-1}\cdots a_1 a_0$. Then the value of A is

$$\boxed{A = A_1 \times 2^n + A_0} \quad (3)$$

while the division equation gives

$$\boxed{A = B \times Q + R} \quad (4)$$

Combining (3) and (4) we get

$$\boxed{A_1 \times 2^n + A_0 = B \times Q + R} \quad (5)$$

Assume now that $A_1 \geqslant B$ and assume that division overflow doesn't occur. The assumption of no overflow occuring means that Q can be represented in n bits or $Q_{maximum} = 2^n - 1$. Under this assumption, the maximum of the right side of equation (5) is (as a function of B)

$$(B \times Q + R)_{\text{maximum}} = B \times Q_{\text{max}} + R_{\text{max}} = B \times (2^n - 1) + B - 1$$
$$= B \times 2^n - B + B - 1 = B \times 2^n - 1$$; (we took into account the fact that $R < B$ which means $R_{\text{max}} = B - 1$).

To summarize we have

$$\boxed{\text{Maximum of right side of eq (5)} = B \times 2^n - 1} \qquad (6)$$

Let's now find the minimum of the left side of equation (5) under the assumption that $A_1 \geqslant B$. Under this assumption $A_{1\,\text{minimum}} = B$. So we now have

$$(A_1 \times 2^n + A_0)_{\text{minimum}} = A_{1\,\text{min}} \times 2^n + A_{0\,\text{min}} = B \times 2^n + 0 = B \times 2^n$$

or

$$\boxed{\text{Minimum of left side of eq (5)} = B \times 2^n} \qquad (7).$$

From equations (6) and (7) we get

min. of left side of eq (5) $>$ max of right side of eq (5)

This last fact simply means that if we assume an n-bit quotient Q, the maximum of the right side of eq (5) can't even reach the minimum of the left side of eq (5). That means that the quotient Q should be longer than n-bits long which implies a division overflow.

Proof of statement 2:

Here we have to prove that if $A_1 < B$ then division overflow doesn't occur. Consider again equation (5) and assume that no overflow occurs. No overflow occuring means that Q can be represented with n bits or $Q_{maximum} = 2^n - 1$. Under this assumption, the maximum of the right side of equation (5) is again

$$(B \times Q + R)_{max} = B \times Q_{max} + R_{max} = B \times (2^n - 1) + B - 1 = B \times 2^n - 1$$

or

$$\boxed{\text{Maximum of right side of eq. (5)} = B \times 2^n - 1} \quad (8)$$

Looking now at the left side of equation (5) we have $A_{1\,maximum} = B - 1$ (since we assumed that $A_1 < B$) while $A_{0\,maximum} = 2^n - 1$ (since A_0 is n-bit long). As a result, the maximum of the left side of equation (5) is $(A_1 \times 2^n + A_0)_{max} =$

$$A_{1\,max} \times 2^n + A_{0\,max} = (B-1) \times 2^n + 2^n - 1 =$$
$$= B \times 2^n - 2^n + 2^n - 1 = B \times 2^n - 1$$

or

$$\boxed{\text{Maximum of left side of eq (5)} = B \times 2^n - 1} \quad (9).$$

Looking at equations (8) and (9) we can conclude that if $A_1 < B$, then an n-bit quotient is sufficient in order to make the maximum of the right side of eq. (5) equal to the maximum possible value of the left side of eq. (5). Thus there is no need for a longer quotient which means that division overflow doesn't occur.

Example 1: Consider the 8-bit dividend $A = 0111\ 1010$ and the 4-bit divisor $B = 0110$. Do you expect a division overflow to occur if you were to divide A by B?

Answer: Here the left-most 4-bit part of the dividend is $A_1 = 0111 = 7$. The divisor is $B = 0110 = 6$. Since $A_1 > B$ a division overflow will occur. (You can easily see that $A = \underbrace{0111}_{A_1}\ \underbrace{1010}_{A_0} = A_1 \times 2^4 + A_0 = 7 \times 16 + 10 = 122$; $B = 0110 = 6$ and the division of $A = 122$ by $B = 6$ will return a quotient $Q = 20$ and a remainder $R = 2$. The quotient $Q = 20$ requires more than 4 bits for its representation).

Example 2: Consider the 8-bit dividend $A = 0110\ 1010$ and the 4-bit divisor $B = 0110$. Do you expect a division overflow to occur if you were to divide A by B?

Answer: Here the left-most 4-bit part of the dividend is $A_1 = 0110 = 6$ and the divisor is $B = 0110 = 6$. Since $A_1 = B$ a division overflow will occur. (You can easily see that $A = 6 \times 16 + 10 = 106$, $B = 6$ and the division of 106 by 6 will give quotient $Q = 17$ and remainder $R = 4$. The quot. $Q = 17$ requires more than 4 bits for its represent).

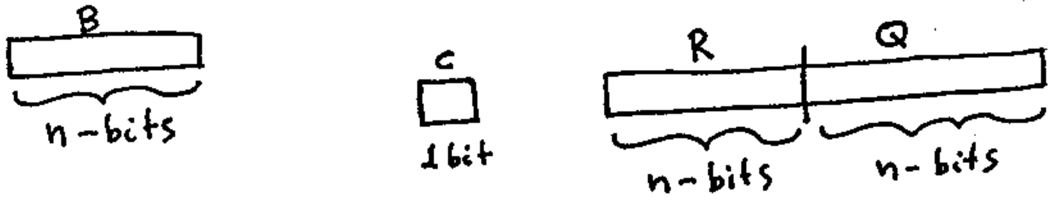
Example 3: Consider the 8-bit dividend $A = 0101\ 1110$ and the 4-bit divisor $B = 0110$. Do you expect a division overflow to occur if you were to divide A by B?

Answer: Here the left most 4-bit part of the dividend is $A_1 = 0101 = 5$ and the divisor is $B = 0110 = 6$. Since $A_1 < B$ division overflow will not occur.

Sequential shift-subtract/add algorithm for binary division

In this section we will present a sequential algorithm for dividing a 2n-bit positive dividend $A = a_{2n-1} \cdots a_1 a_0$ by an n-bit positive divisor $B = b_{n-1} \cdots b_1 b_0$. The dividend A and divisor B are considered to be positive numbers which means that their sign bits are zero (or $a_{2n-1} = 0$ and $b_{n-1} = 0$). Division algorithms involving negative numbers exist but are more complicated so they are not going to be presented here.

Consider the following fields: An n-bit field B; a 2n-bit field R, Q (R is n-bit field; Q is n-bit field); a 1-bit field c. These fields involved in the division are shown below



The sequential division algorithm is now as follows:

- Initialization: Initialize field B with divisor; Initialize field R, Q with dividend (most signif. n-bit part of dividend goes in R while least signif. n-bit part of dividend goes into Q). Initialize the 1-bit field c with anything 0 or 1 (it doesn't actually matter).

•• The shift-subtract/add division algorithm:

After initialization you have to perform n cycles of shift-subtract/add as follows:

— For the <u>first cycle</u> do the following: Shift R, Q one bit to the left. This creates a vacant position at the right-most bit of field Q. Following the shift operation subtract the divisor B from the field R. Place the result of the subtraction in the field R and place the created carry out both in the 1-bit field c as well as in the right-most (vacant) position of the field Q (of course, subtracting B from R means adding the 2's complement of B to R).

— For the <u>remaining n-1 cycles</u> do the following:

If the carry out c from the previous cycle is zero then shift R, Q one bit to the left and following the shift operation add the divisor B to the field R. Place the result of the addition in R and the created carry out in c as well as in the right-most bit of Q.

Else

If the carry out c from the previous cycle is one, shift R, Q one bit to the left and following the shift operation subtract the divisor B from R, place result in R and carry out in c as well as in the right most bit of Q.

••• <u>After the n-th shift-subtract/add cycle:</u>

After the completion of the n-th shift-subtr./add cycle check the carry out c created by this n-th cycle. If this carry out c is one then the division is over and the field R contains the correct remainder while the field Q contains the correct quotient.

If however, the carry out c created by the n-th shift-subtr./add cycle is zero, then the correct quotient Q is found in the field Q but the number in the field R is not the correct remainder. In order to get the correct remainder you have to add the divisor B to the field R and place the result back in R (ignore carry out this time). Now the field R will contain the correct remainder. This final cycle is called <u>restore</u> cycle.

We can now briefly summarize the presented sequential division algorithm.

<u>Summary</u>: If a 2n-bit positive dividend A needs to be divided by an n-bit positive divisor B, the following need to take place:

- Initialize field R, Q with dividend etc ...

•• Perform n cycles of shift-subtract or shift-add. A shift-subtract will take place if the previous cycle resulted in carry out of 1 while a shift-add will take place if the previous cycle resulted in carry out of 0. The very first cycle is a shift-subtract. Each created carry out fills the right most vacant position of the quotient field Q. The shift operations are left shifts.

••• At the end, restoration of the remainder field R is necessary only if the carry out of the last shift-subtr./add cycle is 0. Restoration means adding the divisor B to the field R.

The above algorithm returns the remainder of the division in the field R and the quotiend in the field Q.

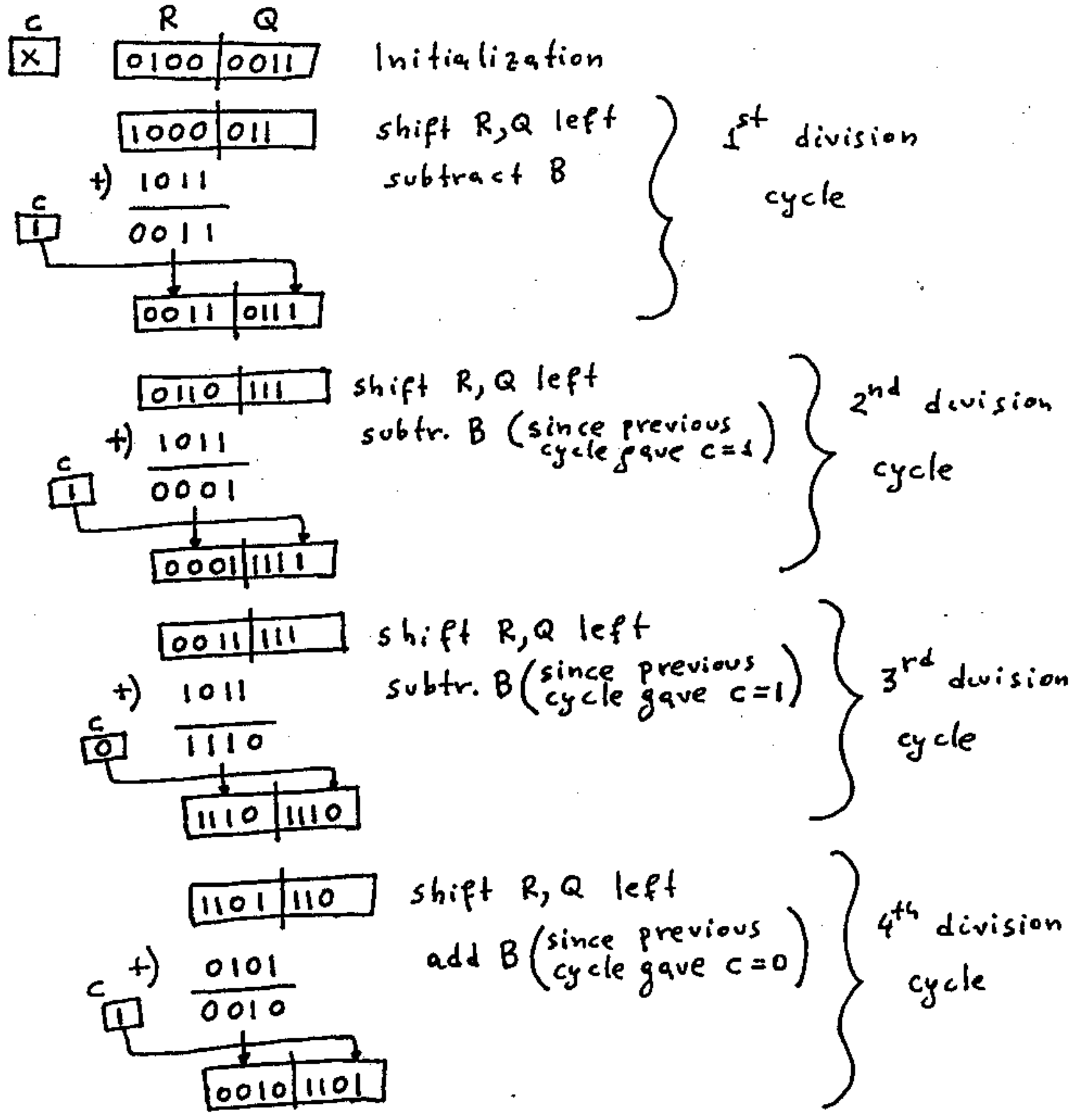
The following examples clarify the sequential division:

<u>Example 4:</u> Perform the division of the 8-bit dividend $A = 01000011$ by the 4-bit divisor $B = 0101$.

<u>Answer:</u>

Here (of course) $n = 4$. Also, the 2's complement of B is

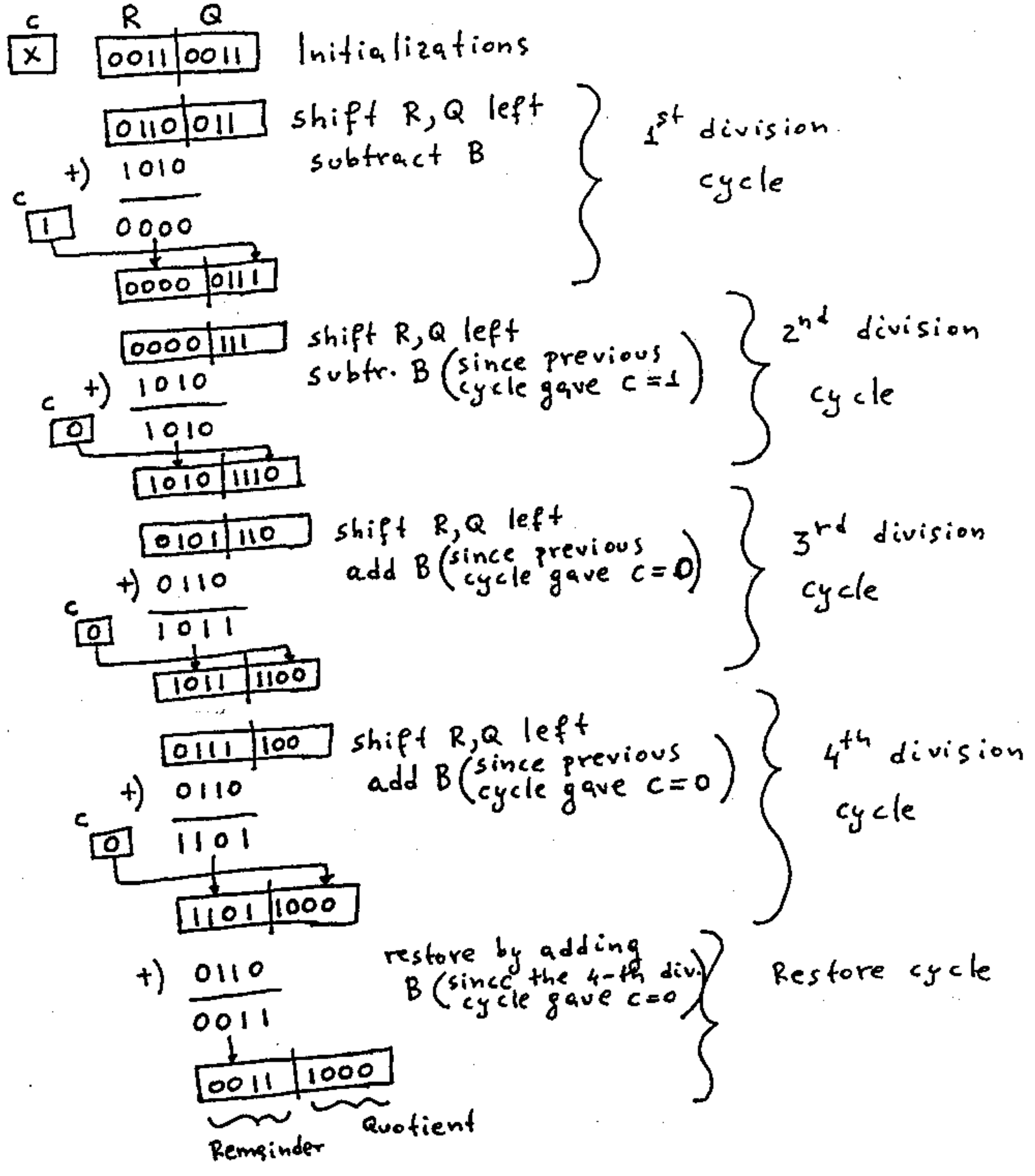
$-B = 2\text{'s compl. } B = 1011$



No restore cycle is necessary since the carry out of the 4th division cycle is 1. Thus R contains the correct remainder $R = 0010$ and Q the correct quotient $Q = 1101$. Double check.

Example 5: Perform the division of the 8-bit dividend A = 00110011 by the 4-bit divisor B = 0110.

Answer: Here n = 4, B = 0110 and −B = 2's compl. of B = 1010



Here we needed a final restore cycle. After restoring we got the correct remainder R = 0011 = 3. The quotient is Q = 1000 = 8. Double check to see that dividing A = 00110011 = 51 by B = 0110 gives Q = 8 and R = 3.