EEE339/336 Solutions Sheet 2 - NJP

Question 1

```
  1011
  1010  x
------
  0000
 10110
000000
1011000
-------
1101110
```

To construct the table, we can note that initially:

i) The multiplicand is loaded into the bottom 4 bits of the 2*n*-bit wide multiplicand shift register. (The upper four bits are zero.)

ii) The multiplier is loaded into the Multiplier shift register.

iii) The 2*n*-bit wide accumulator register is loaded with zero. (This register holds the running total of the partial products.)

Denoting the time instants *just before* the rising clock edges as 1., 2., etc.

Time	Contents of 2*n*-bit wide multiplicand shift register	Multiplier shift register	Output of adder	Contents of 2*n*-bit wide accumulator register
1.	00001011	1010	00001011	00000000
2.	00010110	*101	00010110	00000000
3.	00101100	**10	01000010	00010110
4.	01011000	***1	01101110	00010110
Final	Don't care	****	Don't care	01101110

Therefore, after the fourth clock edge, the accumulator holds the final product.

The contents of the accumulator register holds the running total of the partial products; in operation if the circuit is required to add zero, it does this by leaving the state of the accumulator unchanged.

Question 2

Considering two, 3-digit 2s-complement numbers, A and B, of the form:

$$A = -a_2 2^2 + a_1 2^1 + a_0 2^0 \text{ and } B = -b_2 2^2 + b_1 2^1 + b_0 2^0$$

It follows that the (signed) product must be 6-digits wide and the weightings of each digit in this product must be according to:

$$A \times B = C = -c_5 2^5 + c_4 2^4 + c_3 2^3 + c_2 2^2 + c_1 2^1 + c_0 2^0$$

EEE339/336 Solutions Sheet 2 - NJP

Forming the first partial product from the shift-&-add multiplication:

$$-a_2b_0 2^2 + a_1b_0 2^1 + a_0b_0 2^0$$

where the $a_0b_0 2^0$ term will contribute the LSB of the product, the $a_1b_0 2^1$ contributes to the second least significant digit of the product and the $-a_2b_0 2^2$ contributes to the third least significant. It is immediately apparent that the first partial product which contributes a term to the third least significant digit which has the wrong weight: -2^2 instead of $+2^2$. This problem recurs with every partial product contributing digits to the final product which have the wrong weighting. Consequently, the sum of the partial products will be plain nonsense since some of the terms we are summing will have the correct weighting and some will have the wrong weighting; therefore, the basic shift-and-add method does not work for 2s-complement numbers.

i) Turning to why the basic method fails for a negative multiplicand, it is because the digit weighting of the shifted multiplicand is incorrect. However, if we *sign extend* the shifted multiplicand to the full width of the product, the negative weighting is always confined to the left-most digit of the shifted multiplicand. Hence it is always consistent and will therefore yield the correct answer. Thus, for multiplying -7 x 3, failing to sign extend the shifted multiplicand gives:

1001	x	(-7)
0011		(+3)
00001001		$2^3+1 = +9$
00010010		$2^4+2 = +18$
00000000		
00000000		
00011011		= + 27

This is clearly the wrong answer!

Sign extending the shifted multiplicands to the full width of the double-length product, however, gives:

1001	x	(-7)
0011		(+3)
11111001		
11110010		
00000000		
00000000		
11101011		-128 + 64 + 32 + 8 + 2 + 1 = -21

This is correct!

ii) Shift-and-add multiplication can be further extended to cope with negative multipliers by noting the weighting on the digits in the multiplier, in particular, recognising that the weighting on the *n*-th bit is -2^{n-1}. Therefore, by *subtracting* the shifted multiplicand which originates from the *n*-th multiplier digit maintains the correct weighting of this multiplier digit. Thus:

	0101 x	(+5)
	1011	(-5)
+	00000101	+5
+	00001010	+10
+	00000000	
-	00101000	= 32 + 8 = 40
	11100111	= -25

Thus by subtracting the final shifted multiplicand, the correct answer is obtained. In practice, the 2s-complement of the final partial product would be added.

Question 3

Starting the division process for a divisor left-shift of four digits, entering the contents of the respective registers just before the rising clock edge gives the following table. Notice that the starting left-shift of four digits is a little arbitrary here; you can start with any number of left-shifts (≥2) but this involves more work for little gain. In reality, a hardware circuit calculating an 8-bit quotient would simply perform all eight cycles.

Time (just before rising clock edge)	Contents of dividend register	Contents of divisor shift register	Output of subtractor	Input to quotient shift register
1.	00010111	01100000	10110111	0
2.	00010111	00110000	11100111	0
3.	00010111	00011000	11111111	0
4.	00010111	00001100	00001011	1
5.	00001011	00000110	00000101	1

Notice that the left-shift of the divisor is four digits on the first clock edge (corresponding to multiplying the divisor by 2^4), three digits on the second clock edge and so forth. The quotient can be read from the right-most column as 00011 and the remainder is the figure in the dividend register at the end of five clock cycles – 00000101.

EEE339/336 Solutions Sheet 2 - NJP

Question 4

Denoting the divisor as V, setting out the table as follows:

N	Dividend, D	2^n x V	D - 2^n x V	Quotient bit
3	1010	0101000	11100010	0
2	1010	010100	11110110	0
1	1010	01010	0	1
0	0	0101	11111011	0

Notice the result for D - 2^n x V is given in 2s-complement form; it is immediately apparent that the values for n = 3,2,0 are negative (*i.e.* that D - 2^n x V ≥ 0 is false).

The answer is thus: 0010 remainder zero. (As a check: $1010_2 = 10_{10}$ and $0101_2 = 5_{10.}$ Hence the answer should be 2 remainder 0, or 0010_2 remainder 0010_2.)

The operation of the circuit relies on the adder circuit being able to generate a stable sum before the next clock edge which saves the running total to the 2n-bit wide accumulator register. Therefore, the maximum clock rate at which the circuit will work is determined by the propagation delay in the adder.

Question 5

n	Dividend	Shifted divisor	Op	Result	Quotient Bit
4	00001101 (13)	01100000 (96)	-	10101101 (-83)	0
3	10101101 (-83)	00110000 (48)	+	11011101 (-35)	0
2	11011101 (-35)	00011000 (24)	+	11110101 (-11)	0
1	11110101 (-11)	00001100 (12)	+	00000001 (1)	1
0	00000001 (1)	00000110 (6)	-	11111011 (–ve)	0

With byte wide storage locations, assuming that the dividend could be any 8 bit positive value, the divisor (0110 with a sign bit) will need to be shifted a maximum of four places.

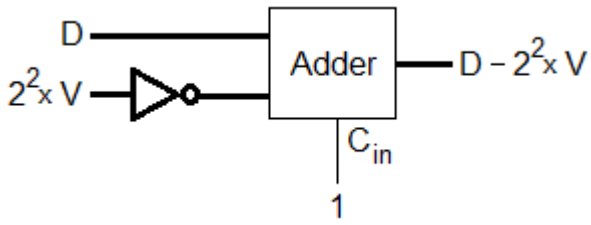
As the final operation has given a negative result, one further addition of 00000110 is made to restore the remainder.

The final result is 00010 remainder 1.

That is, 13 ÷ 5 = 2R1.

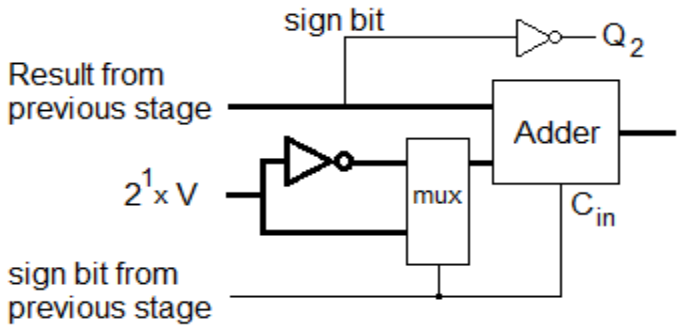
Question 6

The first stage in non-restoring division is to subtract the left-shifted divisor, **V** from the dividend, **D**. This can be easily implemented using a parallel adder, as follows:

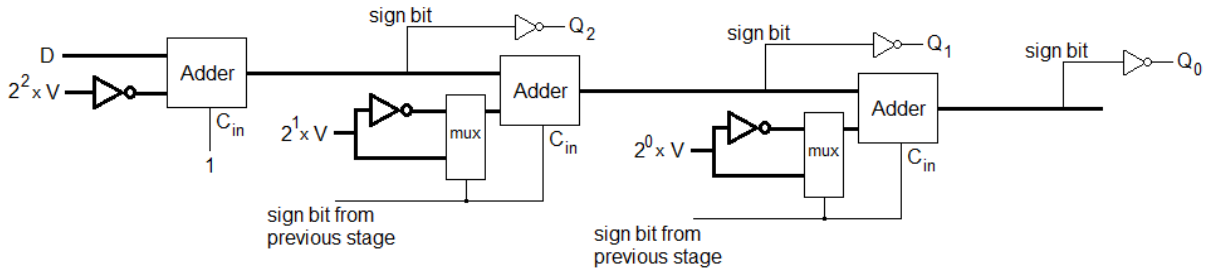


The subtraction is performed by forming the 2s-complement (invert and add 1 via the carry input).

The next stage in non-restoring division depends on the sign of the result of the previous stage; further, the quotient bit we extract $\mathbf{Q_2}$, in this case, also depends on the sign of the result. So the implementation of the second stage needs to either add or subtract depending on the sign of D - 2^n x V above. Assuming the addition/subtraction has been done using 2s-complement numbers, this leads us to a second stage of:



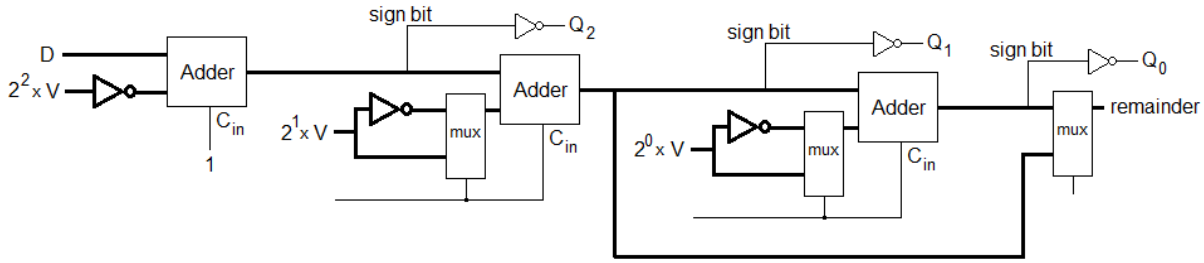
If the sign bit from the previous stage is a 0 indicating a positive number, the mux will select the inverted form of the input and give a carry-in of 1 effectively performing a subtraction by 2s-complement (the inversion required before the C_{in} input is not shown on the diagram). If the sign bit from the previous stage is a 1 indicating a negative number, the mux will select the true form of the input and give a carry-in of 0 effectively performing an addition. The third stage is essentially the same leading to the following circuit:



Notice that the bits of the quotient are made up of the sign bits from each of the addition/subtraction stages.

At the final stage, what emerges from the final adder at the right-hand end of the circuit could be negative – which is meaningless as a remainder! In this case the remainder has to be *restored*. This can be achieved by selecting the remainder from the output of the final adder or the input to the final adder, depending on the sign of what emerges from the final adder.

Thus the final circuit is as follows:



The time to produce a quotient will be dictated by the propagation delays through the three adders. As the width of the operands increases, the length of this chain of adders gets longer and longer which makes the propagation delay proportionately longer.

Question 7

The 2s-complement of a number **a** is given by $\mathbf{2^n - a}$

Taking the 2s-complement of this again gives $\mathbf{2^n - (2^n - a) = 2^n - 2^n + a = a}$