J.Pritchard U1661665

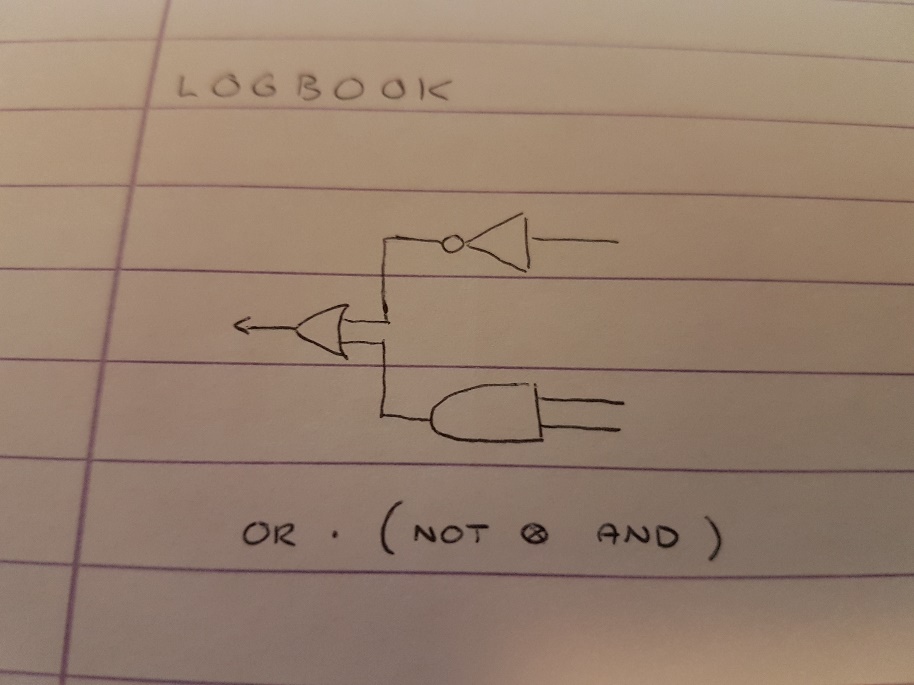
U1661665@unimail.hud.ac.uk

Logbook

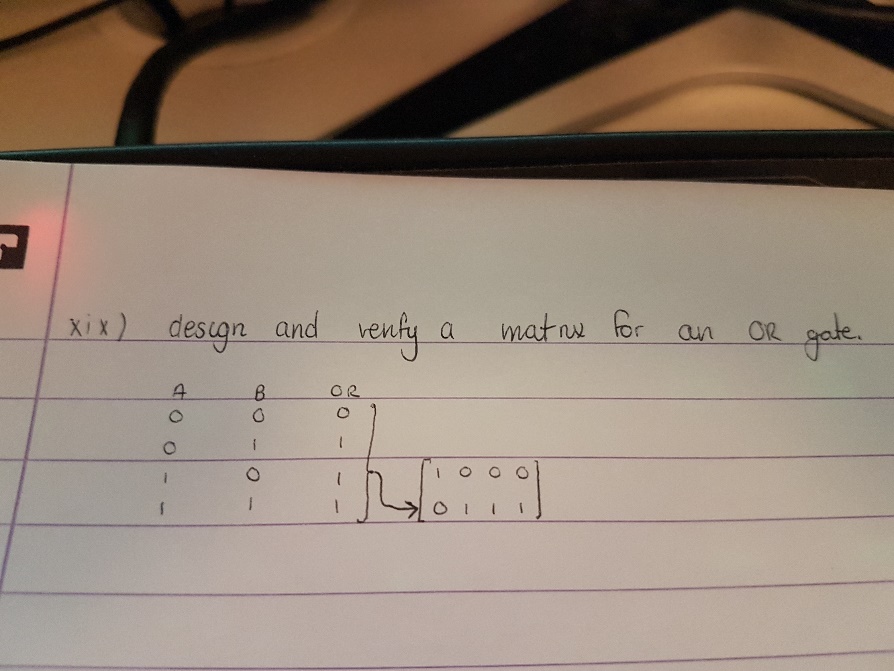
Algorithms – Processes and data

# Practical 13 (Week 17)

## (Logbook) What is the matrix for the following circuit?



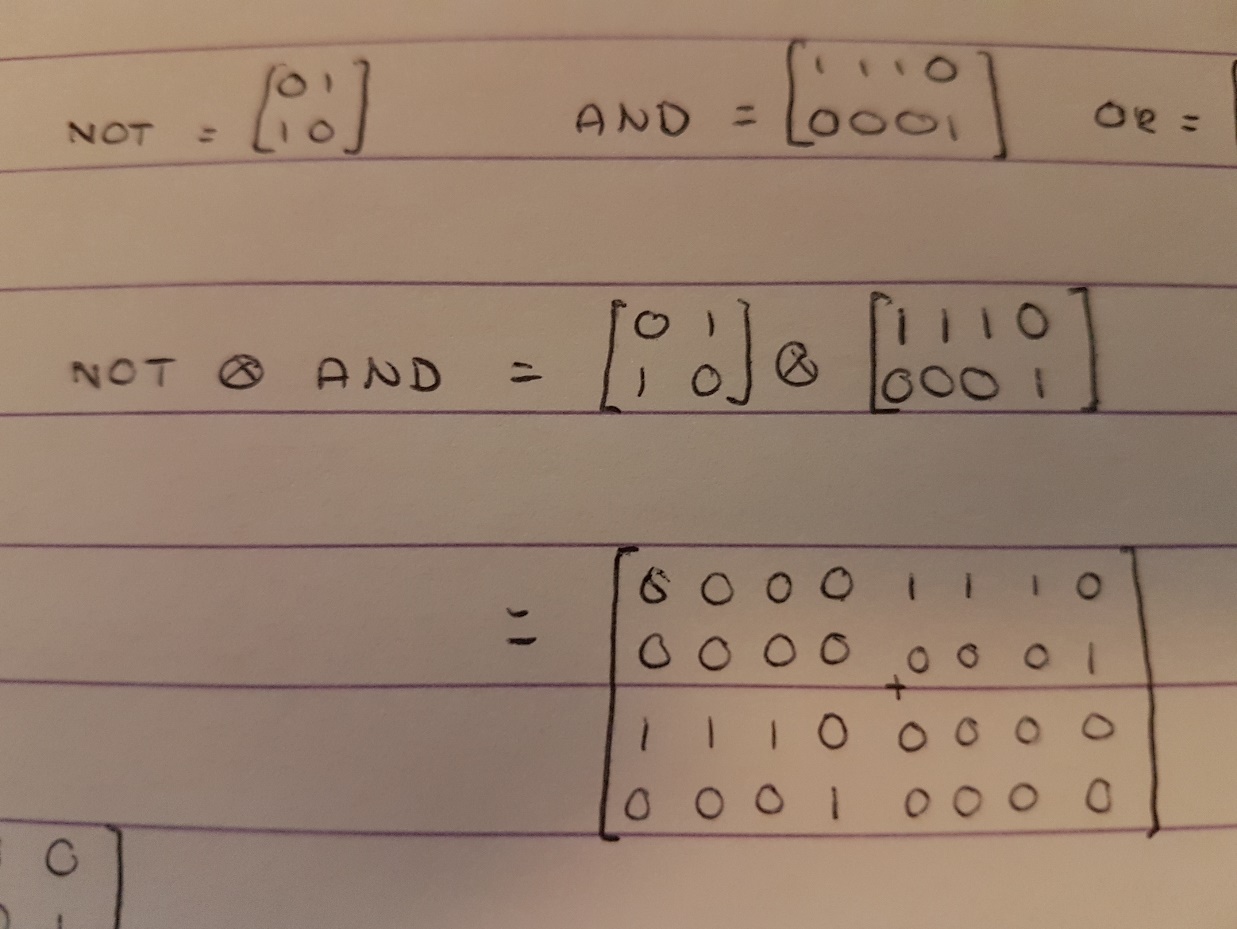
The Logbook is made of a parallel circuit of a NOT & AND, followed by a sequential OR. This is shown by the equation below the diagram.



I was asked previously to design a matrix for an OR gate.

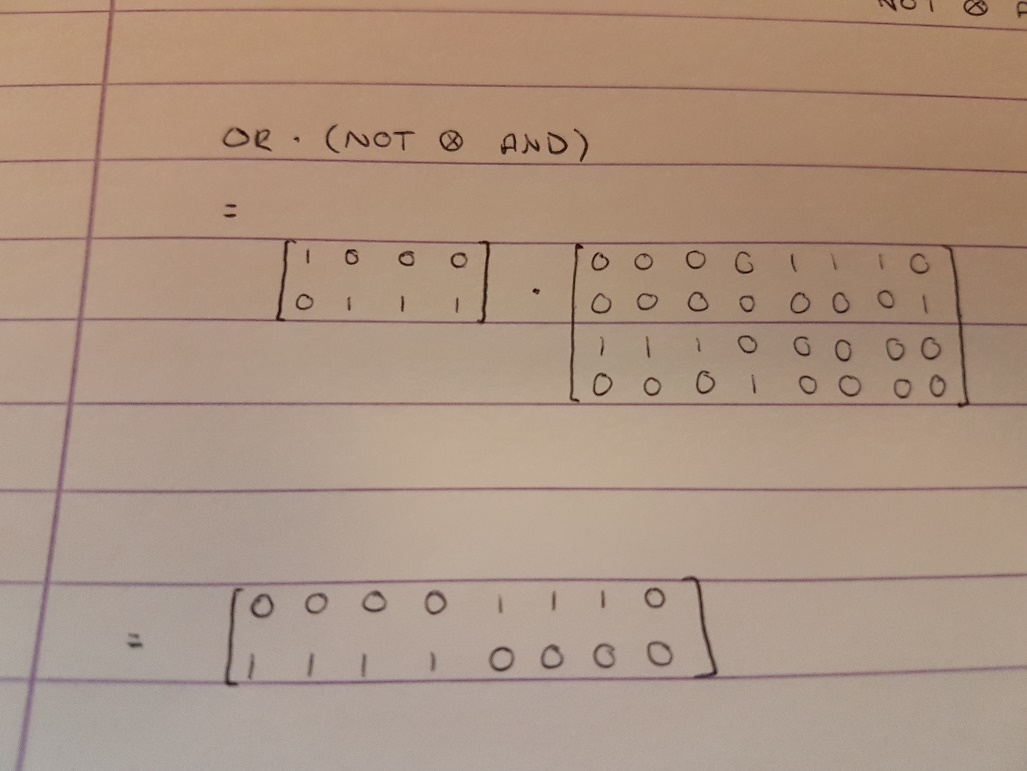
This consists of taking the truth table values for a regular OR gate, putting them top to bottom/left to right, then substituting the values for their individual bit sequence matrices. (1 = [0;1], 0 = [1;0]).

NOTE: the ordering of the 0 and 1 values in the truth table derivation is sensitive here. This must be done in the correct way (as shown) for the resulting matrix to be a correct representation of the gate.

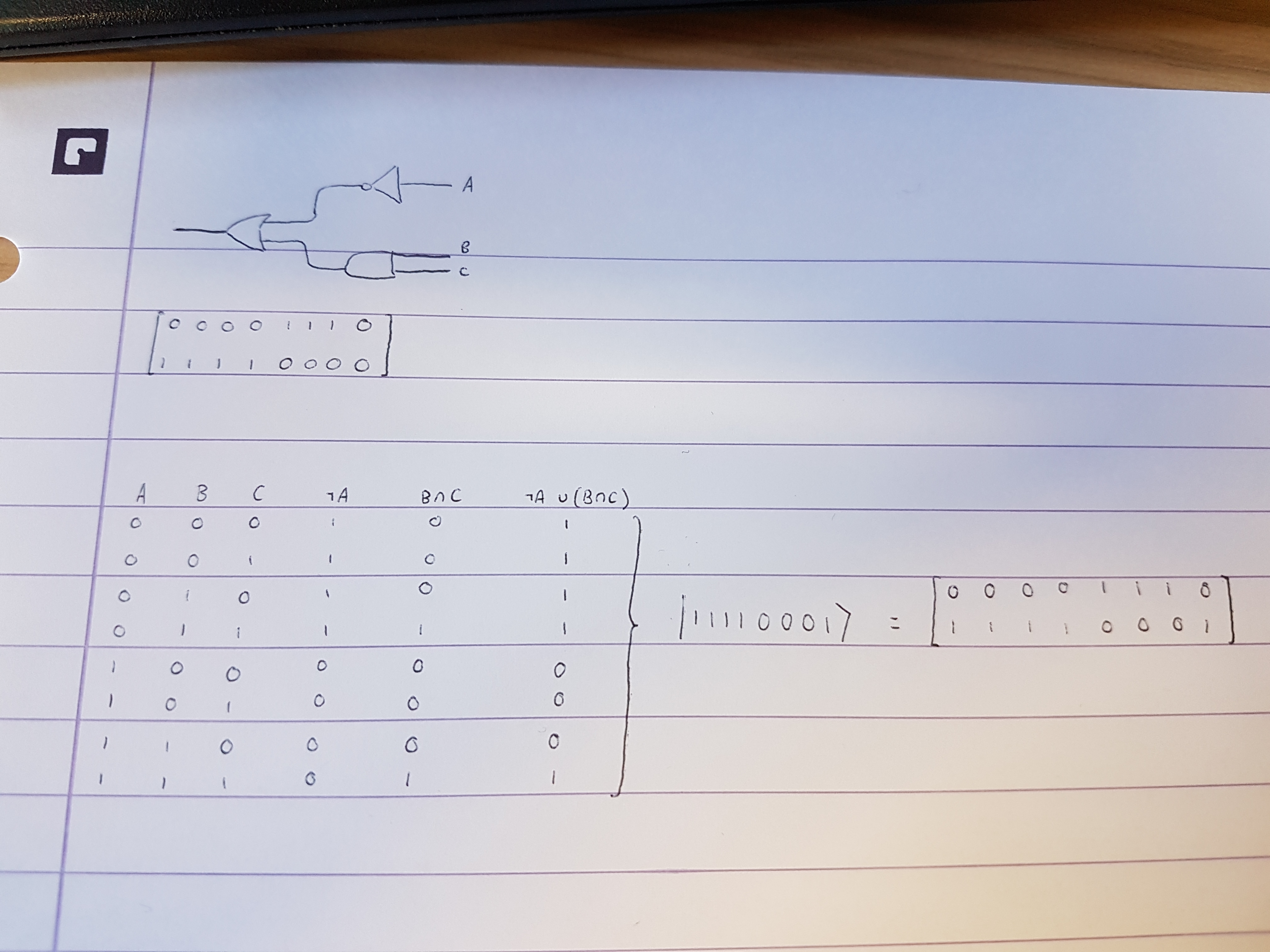


The first step to calculating the matrix for the entire circuit is calculating the parallel component first. This is done by calculating the tensor product of the two gates. Tensor product is calculating the scalar product of each element of matrix A with the entirety of matrix B.

This results in the matrix calculated in the picture.



The result of the previous calculation (NOT ⊗ AND) can then be applied to the sequential OR gate by simply multiplying the two matrices. This results in the complete gate for the circuit.

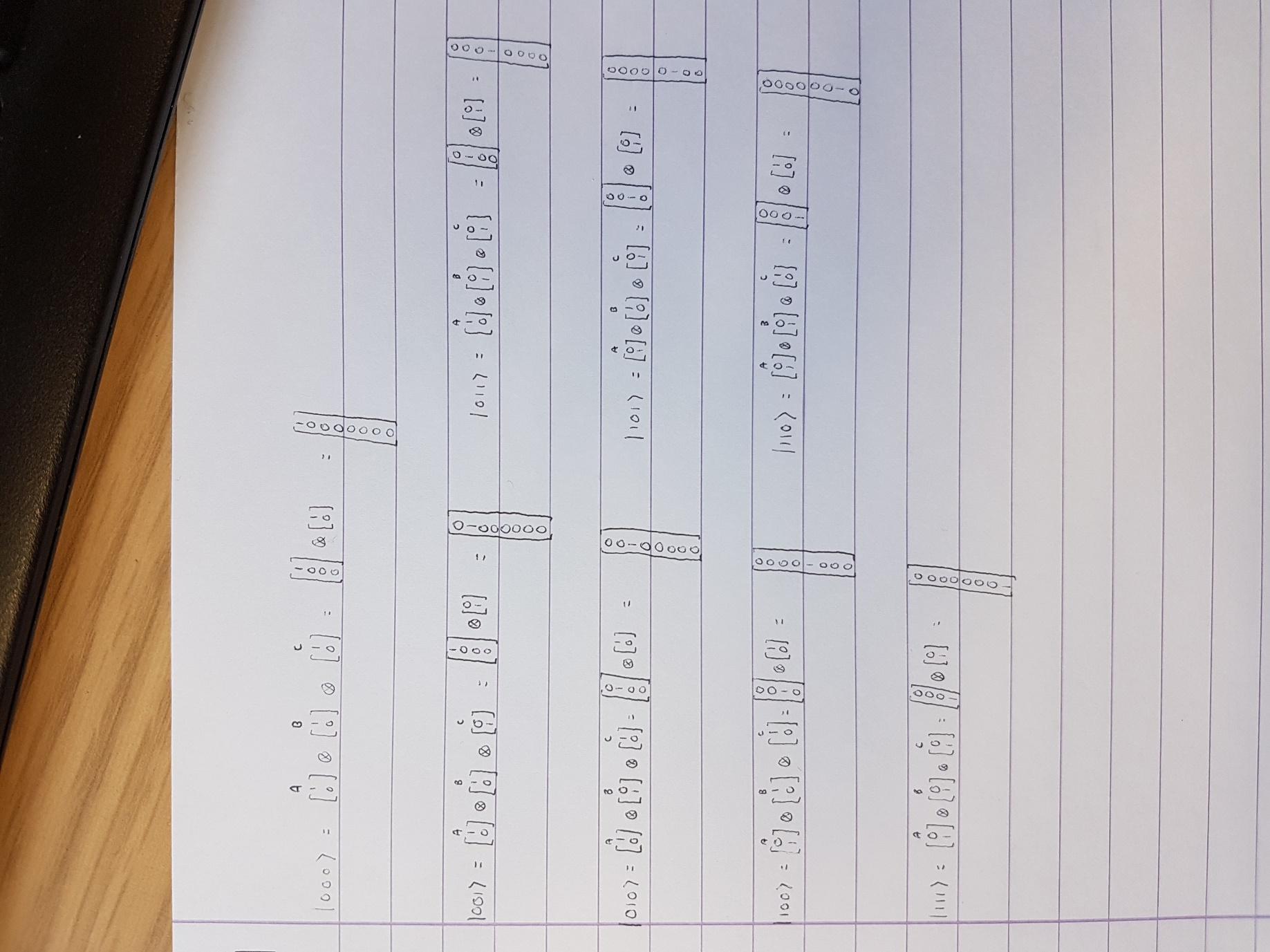


I can verify the Identity Matrix I created for the circuit by working it out with the truth table derivation. In the same way that I used them to derive the matrix for the OR gate, I can use the same ordering method to create a bit stream for the circuit, then turn this into the identity matrix for the circuit.

The two Matrices match, therefore the matrix I derived must be the identity matrix for the circuit.

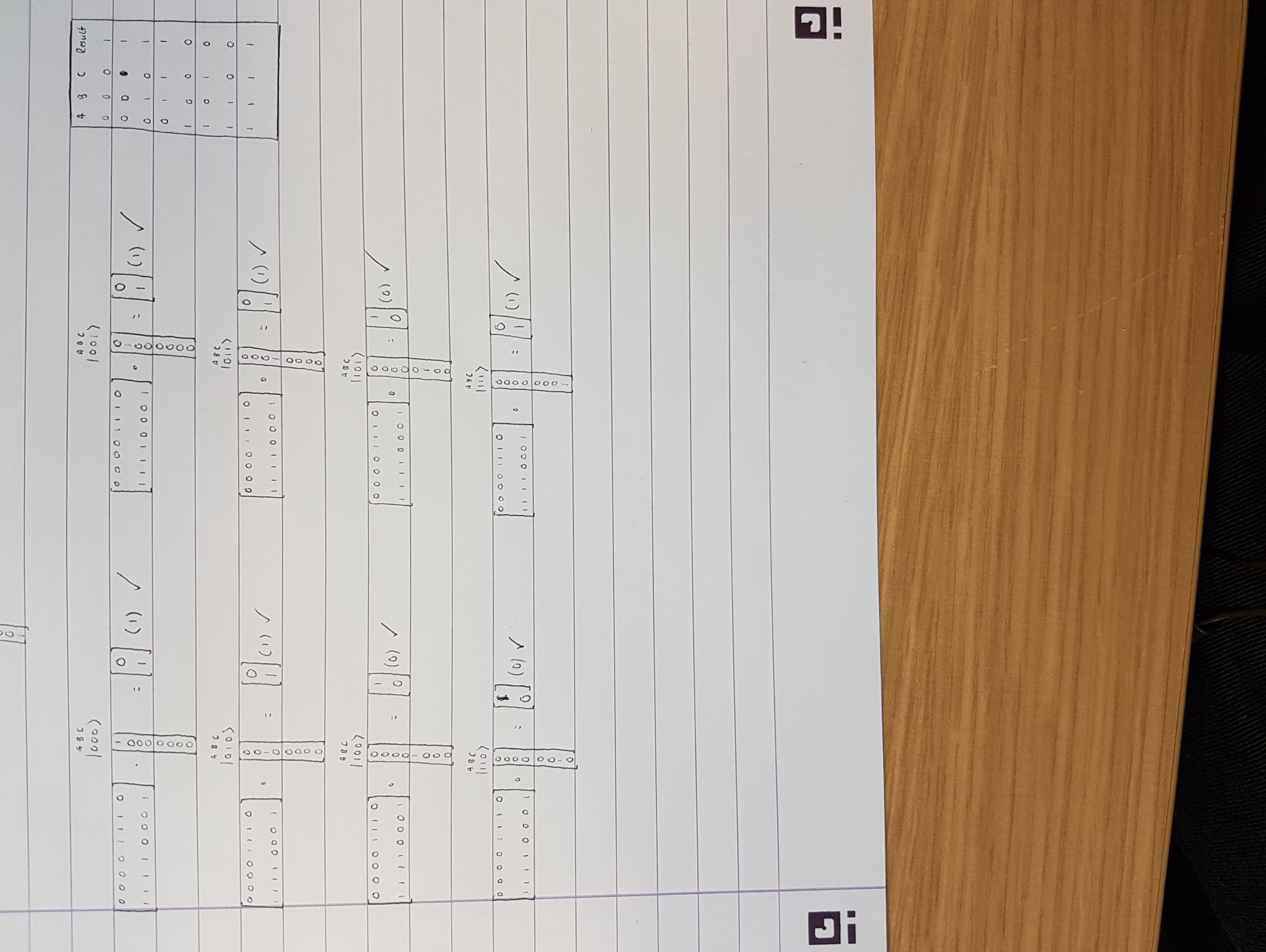
Additionally, there is some testing that can be done to the matrix.

The first step to this is calculating the bit stream matrices for each of the 8 (23) bit streams possible for the truth table/input to the circuit matrix.



Once these have been calculated they can be applied to the circuit matrix via matrix multiplication. This will produce a matrix for a bit of either 1 or 0.

This can be compared with the result for the according bit stream in the truth table. If it matches, then the circuit matrix is a match for that particular bit stream. The circuit matrix can be considered completely correct if every case is a match. The picture below shows that my constructed matrix is a perfect match.



## Self Evaluation.

For 3/5 marks this week I was asked to create a generally correct matrix model for the specified circuit. I have done this.

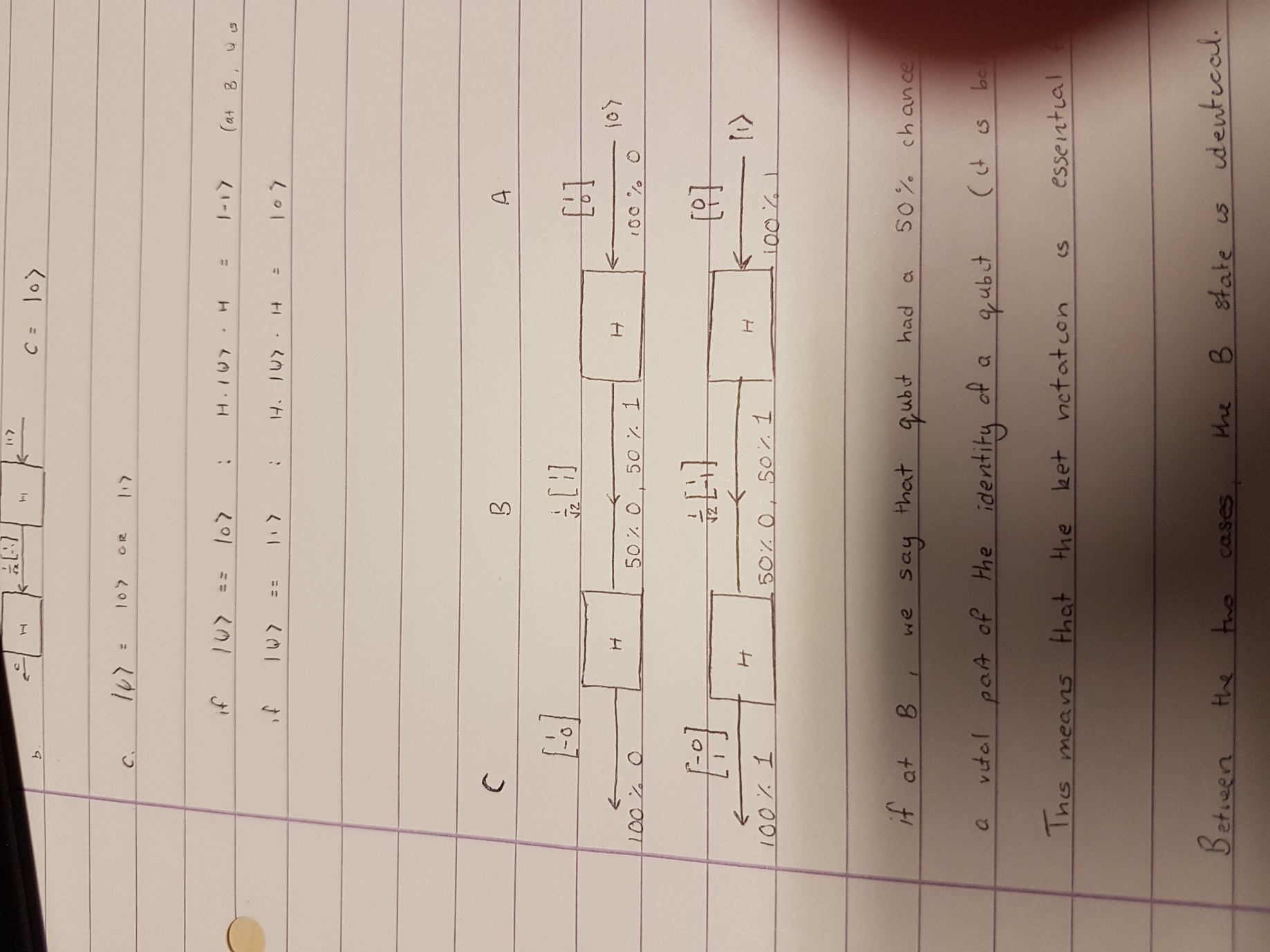
For (4/5)/5 marks I was asked to both explain the derivation and test the model. Throughout my creation I have explained each step of the construction of the matrix and provided verification and testing after the result was obtained.

For the reasons above, I believe my work this week is worth the full 5 marks available.

# Practical 14 (Week 20)

## (Logbook) Question 1.

If |**Ψ**> is a pure state (|**Ψ**> == |0> OR |**Ψ**> == |1>, what happens if |**Ψ**> is passed as an input to a circuit consisting of two Hadamard gates in sequence?



The above picture demonstrates the results of each possibility along with their probabilistic notations below the C, B and A areas.

As is visible, at the B state, the probabilistic notation’s are identical. However, the C states are not. Changing the order of 1 and 0 in the 50/50 notation doesn’t change what they are.

Therefore it is impossible to determine (using the probabilistic model) which C state the 50/50 will de-cohere to.

Now, observing the ket model, the sign on Q­12 is different between the two. This makes it possible to identify which initial qubit was passed into the Hadamard gate (therefore also what it will de-cohere to).

## Self Evaluation.

For 3/5 I was asked to provide a full set of matrices for values at A, B and C which I have done so.

For 4/5 marks I was asked for a well presented working for realising the matrices de-cohere to their original values. The signs do not matter, so I believe I’ve done this.

Finally for 5/5 marks I was asked for understanding that the probabilistic model makes reversibility impossible, which I have both realised and explained.

For the reasons stated above I believe my work is worth the full 5/5 marks this week.

# Practical 15 (Week 21)

## (Logbook) Question 1.

The algorithm shown below will calculate *n*3.

**public static int** cube(**int** n) {

**int** i=0, cube = 0, threeNsq = 0, threeN = 0;

**while** (i *<* n) {

cube = cube + threeNsq + threeN + 1;

threeNsq = threeNsq + 2 threeN + 3;

threeN = threeN + 3;

i++;

}

**return** cube;

}

Prove that the programme above is correct. Add assertions to this code, documenting your proof.

### Code Listing.

package arithmetic;  
  
import javax.swing.JOptionPane;  
  
public class Cube {  
 public static int cube(int n) {  
 int cube = 0, threeNsquared = 0, threeN = 0;  
 int i = 0;  
 assert cube == i\*i\*i : cube;  
 assert threeNsquared == 3\*i\*i : threeNsquared;  
 assert threeN == 3\*i : threeN;  
  
 while (i < n) {  
 assert cube == i\*i\*i : cube;  
 assert threeN == 3\*i : threeN;  
 assert threeNsquared == 3\*i\*i : threeNsquared;  
  
 cube = cube + threeNsquared + threeN + 1;  
 assert cube == (i+1)\*(i+1)\*(i+1) : cube;  
 assert threeN == 3\*i : threeN;  
 assert threeNsquared == 3\*i\*i : threeNsquared;  
  
 threeNsquared = threeNsquared + 2\*threeN + 3;  
 assert cube == (i+1)\*(i+1)\*(i+1) : cube;  
 assert threeN == 3\*i : threeN;  
 assert threeNsquared == 3\*(i+1)\*(i+1) : threeNsquared;  
  
 threeN = threeN + 3;  
 assert cube == (i+1)\*(i+1)\*(i+1) : cube;  
 assert threeN == 3\*(i+1) : threeN;  
 assert threeNsquared == 3\*(i+1)\*(i+1) : threeNsquared;  
  
 i++;  
 assert cube == i\*i\*i : cube;  
 assert threeN == 3\*(i) : threeN;  
 assert threeNsquared == 3\*(i)\*(i) : threeNsquared;  
 }  
 assert i == n : i;  
 assert cube == i\*i\*i : cube;  
 assert cube == n\*n\*n : cube;  
 return cube;  
 }  
   
 public static void main(String[] args) {  
 int x = ReadInt.*read*("Please input a number to be cubed");  
 int result = *cube*(x);  
 JOptionPane.*showMessageDialog*(null,x + "^3=" + result,"Function result",JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
}

### Assertions.

**public static int** cube(**int** n) {

**int** i=0, cube = 0, threeNsq = 0, threeN = 0;

1 : {cube == i3}

2 : {threeNsq == 3i2}

3 : {threeN == 3i}

**while** (i *<* n) {

4 : {cube == i3}

5 : {threeN = 3i}

6 : {threeNsq = 3i2}

cube = cube + threeNsq + threeN + 1;

7 : {cube == (i+1)3}

8 : {threeN == 3i}

9 : {threeNsq == 3i2}

threeNsq = threeNsq + 2 threeN + 3;

10 : {cube == (i+1)3}

11 : {threeN == 3i}

12 : {threeNsq == 3(i+1)2}

threeN = threeN + 3;

13 : {cube == (i+1)3}

14 : {threeN == 3(i+1)}

15 : {threeNsq == 3(i+1)2}

i++;

16 : {cube == i3}

17 : {threeN = 3i}

18 : {threeNsq == 3i2}

}

19 : {I == n}

20 : {cube == i3}

21 : {cube == n3}

**return** cube;

}

### Justifications.

1 : cube should be equal to i3 here, as they are both 0 🡪 03 = 0.

2 : threeNsq should be equal to 3i2 here, as they are both 0 🡪 3(0)2 = 0.

3 : threeN should be equal to 3i here, as they are both 0 🡪 3(0) = 0.

4 : At every step around the while loop, cube will be equal to i3 🡪 cube = cube + threeNsq + threeN + 1;

5 : At every step around the while loop, threeN will be equal to 3i 🡪 threeN = threeN + 3;

6 : At every step around the while loop, threeNsq will be equal to 3i2 🡪 threeNsq = threeNsq + 2threeN + 3;

7 : Cube is increased by one as per the step around the while loop 🡪 i+1 therefore i3 becomes (i+1)3

8 : threeN has not been modified by this point. 🡪 Follows from 5 :

9 : threeNsq has not been modified by this point. 🡪 Follows from 6 :

10 : Cube has not been modified again by this point. 🡪 Follows from 7 :

11 : threeN has not been modified by this point. 🡪 Follows from 8 :

12 : threeNsq gains another addition as per the i step 🡪 3i2 therefore becomes 3(i+1)2

13 : Cube has not been modified again by this point 🡪 Follows from 10 :

14 : threeN gains another addition as per the i step 🡪 3i therefore becomes 3(i+1)

15 : threeNsq has not been modified again by this point 🡪 Follows from 12 :

16 : cube is now only i3 as i has been incremented. 🡪 (i+1) – 1 = i

17 : threeN is now only 3i as i has been incremented. 🡪 (i+1) – 1 = i