**Question 1** (10 points)

A multiplication between two matrices *A* and *B* results in a third matrix *C* where the element in row *i* and column *j* is computed as follows:

This operation can also be applied to bit matrices, i.e., matrices where each element can only assume the values 0 and 1 by substituting additions with XORs and multiplications with ANDs, i.e.:

**Note:** the sum of the products is performed using XORs as well.

Write a routine in ARM assembly called bitMatrixMultiplication that:

1. Takes as parameters the address of 3 bit matrices *A*, *B*, and *C* of size 8 x 8.
2. Computes the product *C* = *AB*
3. Returns *C*

*A*, *B* are stored as vector of 8 bytes in a READONLY area of memory, while *C* is stored as a vector of 8 bytes in a READWRITE area of memory.

For example:

matrixA DCB 0x20, 0x3F, 0xC8, 0x4D, 0x76, 0x58, 0x48, 0x50

matrixB DCB 0Xf8, 0x7C, 0x3E, 0x1F, 0x8F, 0xC7, 0xE3, 0xF1

Keep in mind the following information:

* Each row of the matrix is stored in a single byte
* The bits in each row of the matrix are ordered left-to-right from most to least significant bit

Suggestion.

You can generate 8 bytes, with the value of the bits in each columns of *B*. Then, compute the AND between the whole *i*-th row of A and the *j*-th column of B. Finally, compute the XOR between each bit of the previous result.

Example of computation of the first row of C.

*a0* AND *b0*= 00100000 AND 10001111 = 00000000

*c00* = 0 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 = 0

*a0* AND *b1*= 00100000 AND 11000111 = 00000000

*c01* = 0 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 = 0

*a0* AND *b2*= 00100000 AND 111000011 = 00100000

*c02* = 0 XOR 0 XOR 1 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 = 1

*a0* AND *b3*= 00100000 AND 11110001 = 00100000

*c03* = 0 XOR 0 XOR 1 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 = 1

*a0* AND *b4*= 00100000 AND 11111000 = 00100000

*c04* = 0 XOR 0 XOR 1 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 = 1

*a0* AND *b5*= 00100000 AND 01111100 = 00100000

*c05* = 0 XOR 0 XOR 1 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 = 1

*a0* AND *b6*= 00100000 AND 00111110 = 00100000

*c06* = 0 XOR 0 XOR 1 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 = 1

*a0* AND *b*7= 00100000 AND 00011111 = 00000000

*c07* = 0 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 XOR 0 = 0

Important notes:

1. Write your code inside the “ARM” directory
2. The assembly subroutine must comply with the ARM Architecture Procedure Call Standard (AAPCS) standard (in terms of parameter passing, returned value, callee-saved registers).

**Question 2** (8 points)

Extend the previous exercise as follows.

1. Define three empty vectors of 8 unsigned chars where to store *A*, *B*, and *C*.
2. Initialize TIMER1 to count to 0xFFFF. When the counter reaches 0xFFFF, the timer is reset and no interrupt is generated.
3. When INT0 is pressed:
   1. Read the value of the timer counter of TIMER1.
   2. If INT0 has been pressed less than 8 times, store the most significant byte of the value you just read into *A* and the least significant byte into *B*. The first time store them in the first row, the second time in the second row and so on.
   * **Remember that the timer counter is 32 bits long: you must take into account only on the least significant 16 bits!** If the value you read from the counter is 0x000042AA, the most significant byte is 0x42 and the least significant is 0xAA.
   1. If INT0 has been already pressed 8 times, hence the matrices are full, do nothing.
4. Use KEY1 to call bitMatrixMultiplication. When KEY1 is pressed:
   1. If INT0 has been pressed less than 8 times, do nothing since the matrices have not been filled yet.
   2. If INT0 has already been pressed 8 times, hence the matrices are full, pass *A* and *B* to bitMatrixMultiplication.
   3. Configure TIMER0 to show each row of on the LEDs for 0.5 seconds each.
   4. Turn off the LEDs

The prototype of the function in C language is:

void bitMatrixMultiplication(unsigned char \*A, unsigned char \*B, unsigned char \*C);

**Note:** uint8\_t (defined in stdint.h) is a type alias for unsigned char. If you prefer to use one instead of the other, it is ok: they both represent the same type.