Here's first our gates that is defined in the problem

```
X = matrix([[0,1],[1,0]])
S = matrix([[1,0,0,0], [0,0,1,0], [0,1,0,0], [0,0,0,1]])
TOF = matrix([[1, 0, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 1
```

Furthermore, here's all the identities we are going to use:

```
I2 = matrix.identity(2)
I4 = matrix.identity(4)
I8 = matrix.identity(8)
I16 = matrix.identity(16)
I32 = matrix.identity(32)
```

And now, we will create the components of our expression:

```
G1 = I8.tensor_product(TOF).tensor_product(I4)
G2 = TOF.tensor_product(I2).tensor_product(TOF).tensor_product(I2)
G3 = I4.tensor_product(X).tensor_product(I32)
G4 = I4.tensor_product(TOF).tensor_product(I8)
G5 = I2.tensor_product(TOF).tensor_product(I16)
G6 = I32.tensor_product(S).tensor_product(I2)
G7 = I16.tensor_product(S).tensor_product(S)
G8 = I4.tensor_product(S).tensor_product(I2).tensor_product(S).tensor_product(I2)
G9 = I2.tensor_product(S).tensor_product(I32)
G10 = S.tensor_product(S).tensor_product(X).tensor_product(I8)
G11 = I16.tensor_product(S).tensor_product(I4)
G12 = I8.tensor_product(S).tensor_product(I8)
gates = [G1, G2, G3, G4, G5, G6, G7, G8, G9, G10, G11, G12]
```

1 | **Check matrix size**

In order to do this, we check the size of every single matrix (both rows and columns), and ensure they all agree to being 256.

```
cols = [i.ncols() for i in gates]
rows = [i.nrows() for i in gates]

# set the components to remove duplicates
cols_set = set(cols)
rows_set = set(rows)

# given all sizes equal, we should have one element that's 256
assert (len(cols_set)==1 and list(cols_set)[0] == 256), "cols mismatch"
assert (len(rows_set)==1 and list(rows_set)[0] == 256), "rows mismatch"
```

We can see that they all agree to being 256.

2 | **Check eigenvalues**

We now find the eigenvalues of the matrices in the gates.

```
eigens = [i.eigenvalues() for i in gates]

# we remove duplicates
eigens = [set(i) for i in eigens]

# and check that they are all -1 and and 1
is_one_negone = [i == set([1,-1]) for i in eigens]
assert sum(is_one_negone) == len(is_one_negone), "not all true"
```

We can see that all eignenvalues of the matricies are both one and negative one.

3 | The Algorithm

We now define M:

```
M = G12*G5*G3*G12*G11*G4*G10*G9*G2*G3*G8*G7*G6*G1
```

And, we define x_1, x_2, x_3. We also define $|d\rangle$ as 0, setting the ancillia cubits:

```
a,b, c,d, e,f = var("a b c d e f")

x1 = matrix([[a],[b]])
x2 = matrix([[c],[d]])
x3 = matrix([[e],[f]])

z1 = matrix([[0],[1]])
z2 = matrix([[0],[1]])
z3 = matrix([[0],[1]])
z4 = matrix([[0],[1]])
z5 = matrix([[0],[1]])

input_cubits = z3.tensor_product(x1).tensor_product(x2).tensor_product(x3).tensor_product(z1).tensor_pr

input_cubits
```

We will now do the computation.

```
output_conjoined = M*input_cubits
output_conjoined
```

Finally, we solve for the sixth cubit.

```
x10, x11, x20, x21, x30, x31, x40, x41, x50, x51, x60, x61, x70, x71, x80, x81 = var("x10 x11 x20 x21 x

s1 = matrix([[x10],[x11]])
s2 = matrix([[x20],[x21]])
s3 = matrix([[x30],[x31]])
s4 = matrix([[x40],[x41]])
s5 = matrix([[x50],[x51]])
s6 = matrix([[x60],[x61]])
s7 = matrix([[x70],[x71]])
```

```
s8 = matrix([[x80],[x81]])

output_solutions = s1.tensor_product(s2).tensor_product(s3).tensor_product(s4).tensor_product(s5).tenso


exprs = [i[0] == j[0] for i,j in zip(output_conjoined, output_solutions)]

# solution = solve(exprs, [x10, x11, x20, x21, x30, x31, x40, x41, x50, x51, x60, x61, x70, x71, x80, x8

solution_func(a,b, c,d, e,f) = exprs
solution_func

truth_table = []
for i in [[0,1],[1,0]]:
    for j in [[0,1],[1,0]]:
for k in [[0,1],[1,0]]:
    truth_table.append(((i,j,k), solution_func(i[0], i[1], j[0], j[1], k[0], k[1])))
```