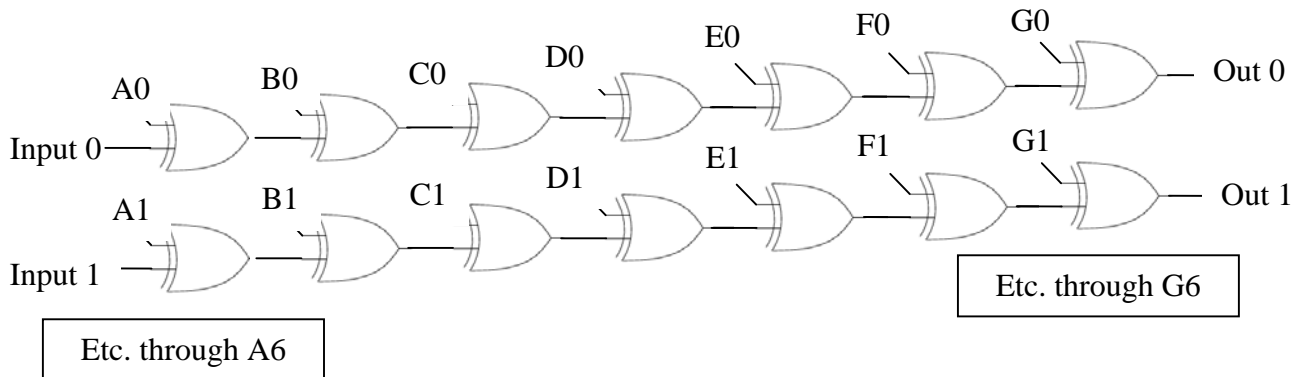


Problem 1: Secret Decoder Ring

Consider the following circuit diagram made up of exclusive-or gates:



Obviously output zero is determined by not only input zero, but also the setting of A_0, B_0, C_0, \dots . Let's call each association between input N and output N a "row". Suppose that there are seven such rows, so that " A_0 " is the upper left external control, " G_0 " is the upper right, " A_6 " is the lower left, " G_6 " is the lower right. One can then specify, in a 7×7 matrix, the setting of all $A_0..G_6$, as in:

1	0	0	1	0	0	1
1	1	1	1	0	0	0
1	1	0	0	0	0	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
1	1	1	0	0	0	0
0	0	0	0	0	0	0

Note that in the case of row six, which is all zeros, this yields an identity function where output six will always be equal to input six.

This matrix and logic can be used as a primitive encoding/decoding device. Take an input character and pass it to the logic diagram with character bit zero going to input zero, bit one going to input one, and so on. Place output zero in bit zero of a new character; place output one in bit one, and so forth. The output will be another character, likely different from the input character.

Example

An easy example can be constructed using the following table:

0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0

In this case bits zero, two, and four will be inverted. Inserting a letter 'A', which is ASCII $0x41$ (hexadecimal) or binary 1000001 will generate 1010100 or $0x54$, which is 'T'. This completes the processing of one character by the Secret Decoder Ring.

After each character is completed, consider each row in the table as an integer. Row zero in the above example has a value of one. Add one to each row after processing a character. Subsequent decoder tables would thus contain:

0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	0	1

After 1st character

0	0	0	0	0	1	1
0	0	0	0	0	1	0
0	0	0	0	0	1	1
0	0	0	0	0	1	0
0	0	0	0	0	1	1
0	0	0	0	0	1	1
0	0	0	0	0	1	0
0	0	0	0	0	1	0

After 2nd character

0	0	0	0	1	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	0
0	0	0	0	0	1	1
0	0	0	0	0	1	0
0	0	0	0	0	1	0
0	0	0	0	0	1	1
0	0	0	0	0	1	1

After 3rd character

Continue this process until the end of the data is encountered (see below) for each table. A row in a table will never overflow a seven bit value, so there is no need to test for this situation.

Although a byte is usually considered as having 8 bits, only the low-order 7 bits are used in this problem. The bit identified as "bit 0" is the low-order bit.

Input

There will be multiple tables and multiple input characters to be decoded using each table. A table will appear first, specified using 7 input lines each containing 7 integers, each of which is either 0 or 1. There is a blank between each pair of bits. Each row is arranged as described above - bit six, then bit five, and so forth down to bit zero. Following the table are additional lines of data, each line representing one character to be decoded. These lines have the same format as the table lines. The input line for the last character to be decoded by a table is followed by a line containing 7 zeroes. This input sequence may then repeat with additional tables and sets of characters to be decoded. The last repetition (a table and characters to be decoded) is followed by a line containing 7 zeroes. (This means no table will ever have 7 zeroes in its first row, and no character to be decoded will consist of 7 zeroes.)

Output

For each case, display the table number (1, 2, ...) and the decoded characters. Display a blank line after the output for each table. Use the format shown in the sample below. There will not be any "non-printable characters" in the output.

Sample Input	Output for the Sample Input
<pre> 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 0 1 0 1 1 1 1 1 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 </pre>	<pre> Table 1: ABCDEFG </pre>