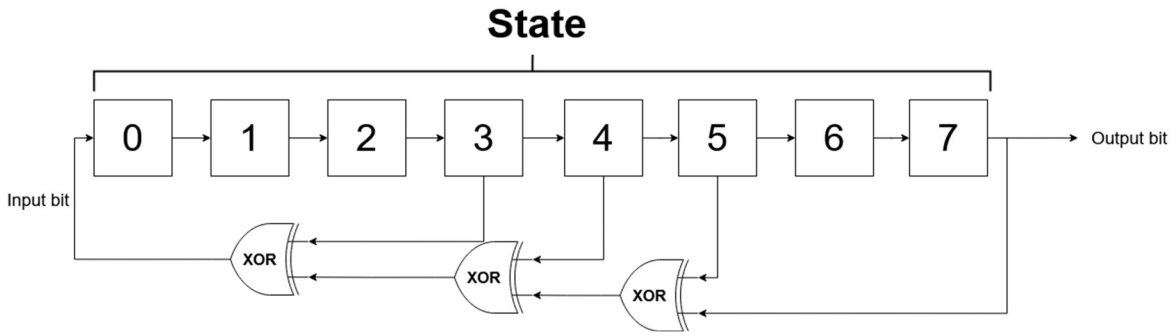
Architetture dei Sistemi di Elaborazione **Laboratory 9**	Delivery date: Until 2AM of 4 December 2024 Expected delivery of lab_09.zip must include: - zipped project folder for Exercise 1.a and 1.b (they should be in the same project)

Exercise 1.a) Implement a 8-bit LFSR

A Linear Feedback Shift Register (LFSR) is a shift register[1] commonly used to implement pseudorandom number generators (PRNGs) in hardware. In LFSRs, the input bit is a linear function of its previous content, usually a XOR between some of the bits of the register.

- The content of the LFSR is called *state.*
- The initial state is called *seed.*
- The next input bit is computed by XORing some of the bits of the current state, called *taps*.
- The next state is computed by shifting the register right by one position and inserting the input bit in the leftmost position. For example, if the taps are bits 3, 4, 5 and 7, the LFSR can be graphically represented as below:



At each clock cycle:

- The input bit is computed using the current value of the taps.
- The state is shifted right by one position, outputting the rightmost bit.
- The input bit becomes the new leftmost bit.

Write a program **in C** that implements a 8-bit LFSR on the LANDTIGER board:

- Initialize the state (an unsigned char) with a seed that **you can choose freely** and show it using the LEDs.
  Note that choosing a seed equal to 0 will result in the LFSR always outputting a 0.
- Each time KEY1 is pressed, call an **assembly** function called `next_state` that receives the current state and the bit position of the taps and returns the next state. The output bit is stored in an integer variable passed by-reference called `output_bit`. The taps are represented by a 8-bit variable containing a 1 in positions corresponding to the position of the taps and 0 elsewhere. For example, for the LFSR above, `taps` contains the value $0x1D_{16} = 00011101_2$
- After `next_state` returns, update the current state and show it on the LEDs.
  The function prototype is:

```
unsigned char next_state(unsigned char current_state, unsigned char taps, int
*output_bit);
```

[1] A shift register is a register which accepts one input bit at a time. The input bit is shifted in from the left, while the rightmost bit is shifted out from the right

The function arguments are:

Type	Name	Description
unsigned char	current_state	The current LFSR state
unsigned char	taps	8-bit long variable containing a 1 in correspondence of the bit position of the taps, 0
int *	output_bit	Pointer to an integer containing the output bit

An example of how `next_state` could look like in C when taps are in positions 3, 4, 5 and 7 is given below:

```
// Compute the output bit
*output_bit = current_state & 1;

// Compute the input bit
// Note that the quantities by which you must shift current_state in order
// to compute input_bit depend on the position of the taps.
// The numbers reported here are just an example.
input_bit = (current_state ^ (current_state >> 2) ^ (current_state >> 3) ^
(current_state >> 4)) & 1;

// Compute the new state
new_state = (current_state >> 1) | (input_bit << 7)
```

For example, if the initial state is 0xAA (10101010 in binary), the next 8 states of the LFSR will be:

1. $11010101_2 = 0xD5_{16} = 213_{10}$
2. $11101010_2 = 0xEA_{16} = 234_{10}$
3. $11110101_2 = 0xF5_{16} = 245_{10}$
4. $11111010_2 = 0xFA_{16} = 250_{10}$
5. $01111101_2 = 0x7D_{16} = 125_{10}$

Exercise 1.b) Experiment with different taps

Since the LFSR state is finite, after a certain number of iterations the LFSR will reach again its initial state and start repeating the same sequence of values. A *m*-bit LFSR is said to be *maximal-length* if the length of the sequence of values before the LFSR reaches the initial state is equal to $2^m - 1$. The length of the sequence depends on number and position of taps. For example, the LFSR of Exercise 1.a is maximal-length: the length of the sequence is $2^8 - 1 = 255$.

Extend the program you wrote in Exercise 1.a by experimenting with number and position of taps and implementing the following features:

- Each time KEY2 is pressed, step the LFSR until it reaches its initial state[2]
- Count the length of the sequence and show it on the LEDs

Fill the table below, indicating the positions of the taps, the length of the sequence and the number of clock cycles needed to reach the initial state:

Tap positions	Length of the sequence	Clock cycles
3, 4, 5, 7	255	2141711
0, 3, 7 (0x91)	127	2123694
0, 7 (0x81)	63	2114670
7 (0x1)	2	2106109

You are welcome to experiment with more than three tap configurations.

Note that poorly-chosen taps may cause a LFSR to reach an all-zeroes state (even with a non-zero initial state) from which it will never exit, resulting in an infinite loop. Since we are not

[2] "Initial state" in this context means the state in which the LFSR was when KEY2 was pressed

requesting you to solve the halting problem, always maintain a tap in position 7, or, if you want to experiment anyway, always check for an all-zeroes state in your loop