SIMD on CPU registers

The Single Instruction Multiple Data (SIMD) paradigm can be implemented at different levels on modern computer machines. In the following exercises, we will implement this paradigm in the registers of our CPU, because some of the basic operations on the registers (such as logic bitwise operations) are performed in parallel on the bits of the registers. In all exercises, we will suppose that our CPU is equipped with 64-bit registers.

Exercise 1 – The Game of Life in 1d

The Game of Life is a *cellular automaton* devised by the British mathematician John Horton Conway in 1970. In the Game of Life, the "world" is generally represented in two dimensions. In this exercise, we will rather consider its one-dimensional variant, where the elements composing the world can occupy a set of contiguous cells topologically organized on a straight line. Every element can potentially have two neighboring elements (for simplicity, we'll suppose that the elements in the first and last cells can have at most one neighbor).

At each evolution step, every cell of the one-dimensional world can be in either of the two following states: occupied by a living element, or empty (emptiness can be consequence of the fact that no element ever occupied the cell, or that the element that was in that cell had meanwhile died). We will consider the following rules for simulating the world evolution in the game:

- if the cell i contains an alive element `'x'`,
  - and its two neighboring cells are occupied by alive elements as well, then this element dies, as by overpopulation, and the cell i is emptied;
  - and its two neighboring cells are both empty, then this element dies, as by underpopulation, and the cell i is emptied;
- if the cell i is empty, and only one of its neighboring cells contains an alive element, then a new element will be added to the cell i, as by reproduction.

Write a C-like code where the entire one-dimensional world is stored in one unique 64-bit register of our CPU. Consider to use two registers, one for the previous generation of the world, and another for the new generation. Temporary registers can also be used to store intermediate information. The only allowed operations in your code are the bitwise logic operations on the registers, as well as bit shift operations.

If you'll initialize your world with cells containing alternating dead and alive elements, the first subsequent generations will look like this:

```
|x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x |
|                                                               x|
|                                                              x |
|                                                             x x|
|                                                            x   |
|                                                           x x  |
|                                                          x   x |
|                                                         x x x x|
|                                                        x       |
|                                                       x x      |
|                                                      x   x     |
|                                                     x x x x    |
|                                                    x       x   |
|                                                   x x     x x  |
|                                                  x   x   x   x |
|                                                 x x x x x x x x|
```

Exercise 2 – Parallel products of integers

We have an array X of unsigned integers, and they all need to be multiplied by the number 15. The array X satisfies the two following properties:

- all integer numbers are strictly smaller than 16;
- the size of the array is a multiple of 8.

Our aim is to optimize the task of multiplying all such unsigned integers by 15 by vectorazing these products on 64-bit registers. Moreover, we will pay particular attention to the complexity of the basic operations that are necessary to implement the products at CPU level, and we will try to replace some of such operations so that the overall complexity is minimized.

Please answer to the following questions in order to be guided towards the solution to this exercise:

1. How many bits do we need to represent every unsigned integer of X?
2. How many unsigned integers of X can fit in a 64-bit register?
3. What is the binary representation of 15? How many bits every integer in X can take after its multiplication by 15? As a consequence, in order to guarantee that no integer overflows any other in the register after the multiplication, how many unsigned integers is advisable to store in one unique 64-bit register in view of this multiplication?
4. When several integers are stored in one register, we cannot use the build-in product operation of the CPU to perform the multiplications by 15. It is therefore necessary to *manually* implement the products via bitwise logic operations. If we implement the standard integer product, how many times we need to invoke an integer adder?
5. But a computer expert whispers in our ears that another implementation of the product by 15 is actually possible where the integer adder is invoked fewer times, by making the overall computation more efficient. Can we decompose the multiplication by 15 in a sequence of sub-operations where some of them can be performed by simpler bitwise operations, instead of always invoking an integer adder? *Hint:* think of multiplying by 2 by shifting one time all bits composing the integer to the left side.
6. If we only consider the complexity of the integer adder (and neglect the complexity of other operations because too little in comparison with the one of the integer adder), what is the relationship between the standard complexity for performing any integer multiplication, and the complexity for this particular multiplication by 15 when implementing the solution you found by answering to the previous question?
7. If X is very very large in size, so that the time required to perform the computations is 1600 minutes on our machine when every product is implemented as "`15*X[i]`", what is the expected computational time when implementing the alternative solution found above? Recall that more than one integer number is processed in parallel in the registers, and that we have reduced the complexity for performing the multiplications.