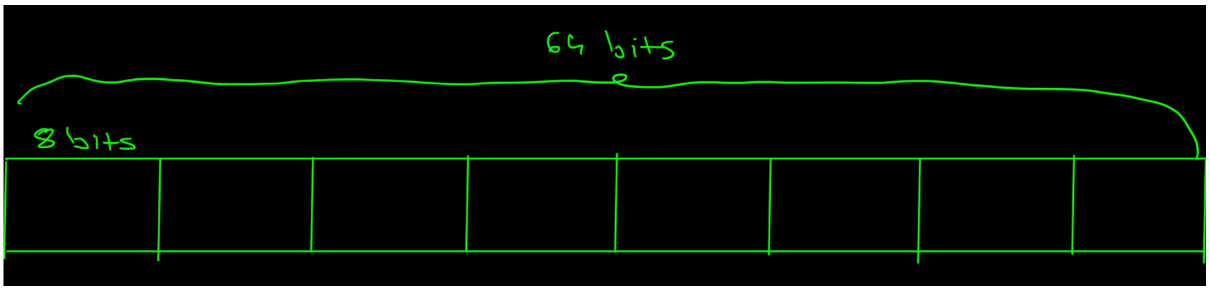
Alright, so, for A5 we need 3 registers:

- ➔ The 1st one has 19 bits
- ➔ The 2nd one has 22 bits
- ➔ The 3rd one has 23 bits
- ⇨ So, 19 + 22 + 23 = 64 bits.

The LFSR will be initialized with an 8 char password. In java, a char has 2 bytes. But this cryptographic algorithm is implemented to work at byte level (so a char should have 1 byte – just like in C/C++). So, in order to get the byte array which is the equivalent of that string, we will use *getBytes()*. And we use characters that are from ASCII table, so that they can be represented on 1 single byte (not characters from UNICODE which require 2 bytes). What I am trying to say is that we can have the initial 8 char password stored in 64 bits. Just like in the picture below:



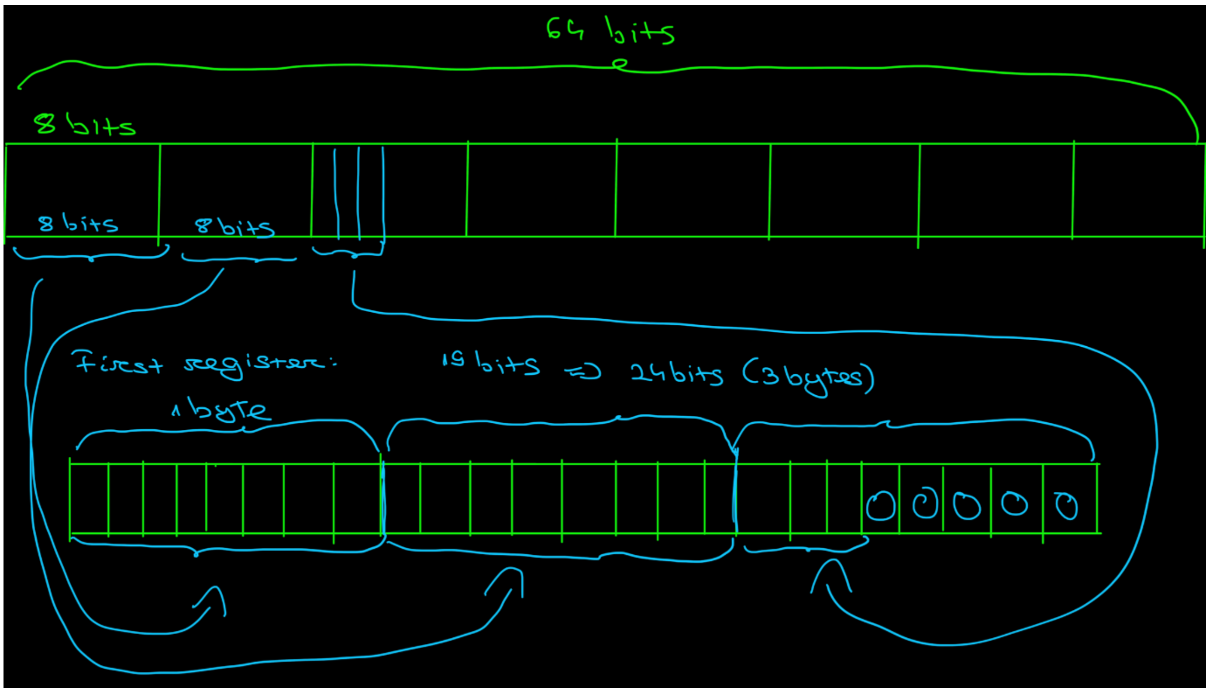
These 64 bits are divided in the following way: 19 + 22 + 23.

So, the fist characters of password will be stored in the first register (19 bits), then in the second register (22 bits) and then in the third register (23 bits). The bits are stored from left to right (just like a normal int[]).

And now we start to solve the problem. This is my perspective on how to solve this assignment.

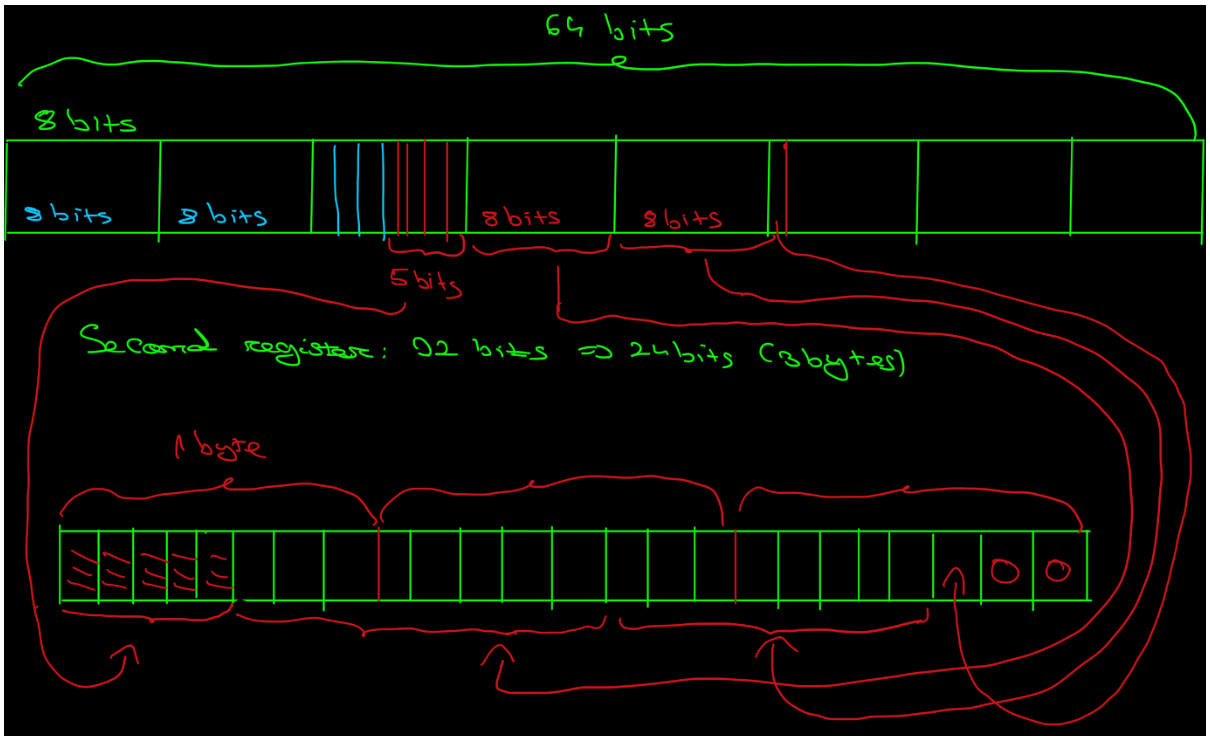
First of all, we must initialize the registers.

The first register is shown in the picture below.

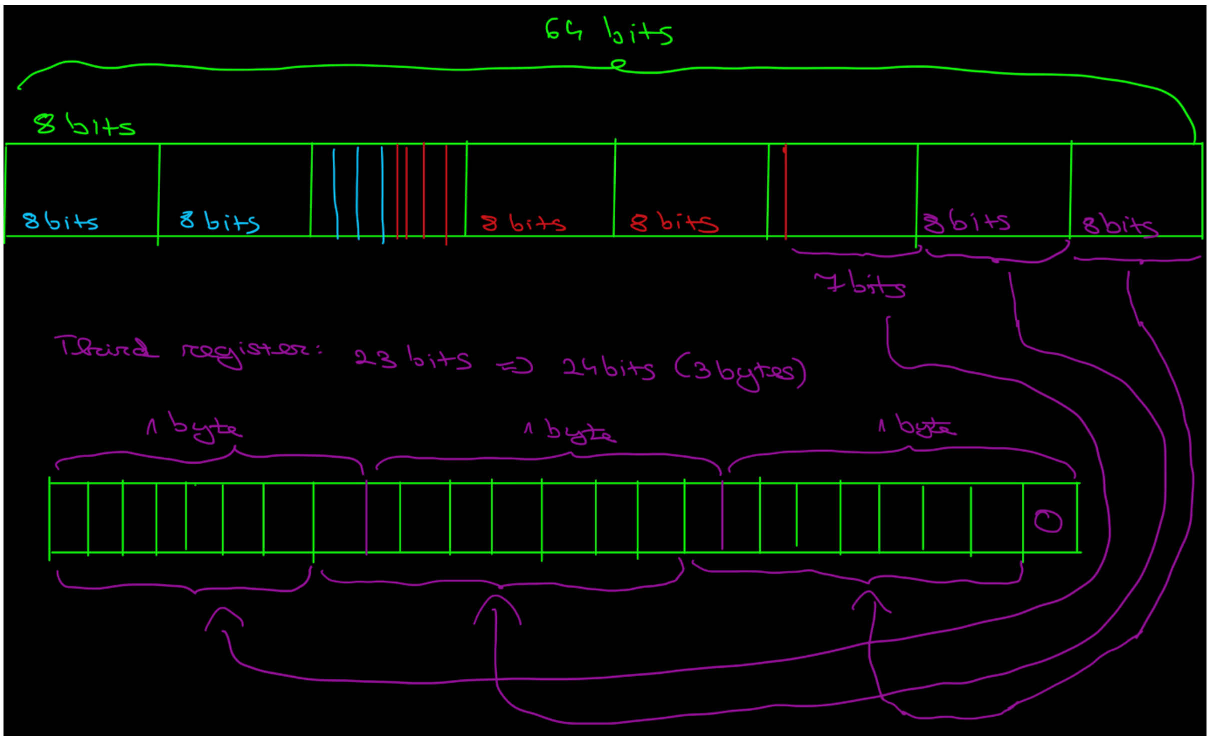


It has 19 bits so we need 24 bits (3 bytes, because we need a byte[] to store bits, we cant store bits as variable type). The first 19 bits of the initial password are stored here. The 5 bits remaining are 0.

The second register is shown in the picture below. The remaining 5 bits from the 3rd byte of the initial password is store at the beginning of the second register. And so on with the next bits. The last 2 bits are 0.



The third register is show below. It has 23 bits, the last bit in 0.



This is how I initialized these 3 registers. This is the way I understood the problem. In my source code, from line 77 to 192 I initialize all these bits.

After that, I just apply the A5 algorithm.

- ➔ XOR between 13, 16, 17, 18 (it starts from 0) in the 1st register => the value is stored in *nextBit1* => this value goes to the beginning of register
- ➔ XOR between 20, 21 (it starts from 0) in the 2nd register => the value is stored in *nextBit2* => this value goes to the beginning of register
- ➔ XOR between 7, 20, 21, 22 (it starts from 0) in the 3rd register => the value is stored in *nextBit3* => this value goes to the beginning of register

Then, XOR between the last value from each register => the value is stored in *randomBitFinal* which is added to the array of random bytes (again, from left to right).

About SHIFTING the registers. Well, giving the fact that the source code quality doesn't really matter now, I wrote 3 functions (each register with its function). In case you are wondering what happens to the zeros that are at the end of each register: well, I just ignore them. Then the shifting algorithm is the one you explain at the course: I store the last bit (which will be XORed later), then store the last bits from the remaining bytes, then shift the bytes, then add the last bits to the beginning of the next.

And this is the A5 from my perspective.

About the other function that generates integers, there is a bug that I can’t fix, I don’t know what is wrong..:(

Thank you for reading this, have a productive day!!