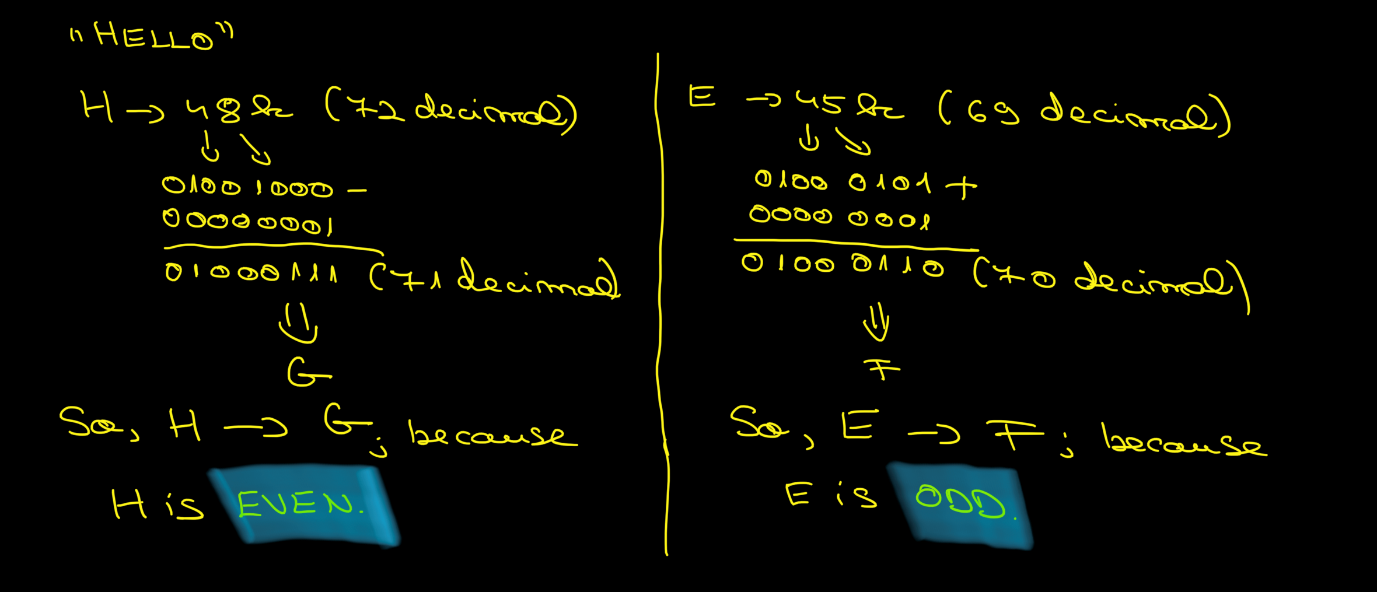
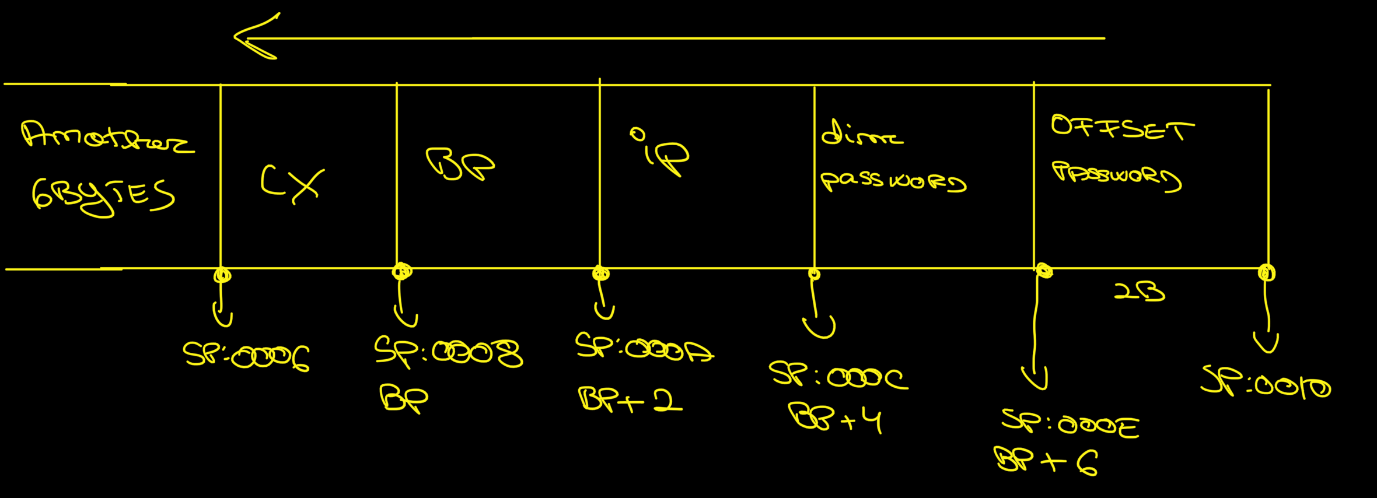
**The second procedure** we use to both encrypt and decrypt the password is called “proc2.asm”. We use a pretty simple algorithm that consists of either adding or subtracting 1b (1 in binary - 0001) depending on the “parity” of each letter. If the letter is even, we add 1, otherwise, we subtract 1. The parity of a letter is given by the letter’s representation in binary. A binary number is odd if the lowest bit is set (for instance: 73 – 0100 1001, the last bit is 1 so it is odd, 72 – 0100 1000, the last bit is 0 so it is even). In the below imagine, this algorithm is illustrated for the letter H (which is even because H – 72 decimal – 0100 1000) and for letter E (which is odd because E – 69 decimal – 0100 0101).



We use *test instruction*1 (test AL, 1b) which checks whether the lowest bit of AL is set.[[1]](#footnote-1) If it is, the number is odd. Otherwise, it is even. Afterwards, we use a *jcc instruction*2 (JZ – j[[2]](#footnote-2)ump if even) in order to either add or substract 1.

We apply this algorithm for all the other characters. A stack is used in order to successfully run this algorithm. In the below picture is drawn the stack. 

In order to decrypt, we apply exactly the same algorithm: if a letter is even, after running the algorithm, it becomes odd and vice-versa. If the algorithm is run again, then the actual letter becomes the initial one (H – 1b = G; G + 1b = H).

1. <https://www.felixcloutier.com/x86/test> [↑](#footnote-ref-1)
2. <https://www.felixcloutier.com/x86/jcc> [↑](#footnote-ref-2)