Bits and Pieces Part 2- Electric Boogaloo

Due: Monday, January 21st

PLEASE READ BEFORE STARTING

Background: All numbers (both integer and float types) that are used in any computer program are always stored in memory (typically RAM) as binary numbers. The range of possible values for these numbers is dependent on how the computer hardware has been constructed (fun fact: the area of research whose focus is on the design and implementation of a set of rules and methods describing the functionality, organization, and implementation of computers is known as computer architecture), and most modern computers have either 32-bit or 64-bit architectures. However, most computing systems (about 98% of microprocessors that are manufactured) are embedded systems, which performs a dedicated function within a larger system. For example, many household appliances such as microwaves and washing machines make use of embedded systems in performing their functions. An important characteristic of embedded systems is that their architectures are much smaller and can typically be 8-bit or 16-bit. To be able to work in such an environment, bitwise operators are utilized in all aspects of embedded systems programming to configure specific registers in performing a specified function.

In this assignment, an 8-bit architecture will be "simulated" in the sense that the values to be passed in the required functions will only be in the range of [0, 255]. Note that the number of possible values that this range contains is exactly $2^8 = 256$.

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0

The above is representative of how an 8-bit register may appear, with BIT7 being known as the most-significant-bit (**MSB**), and BIT0 being known as the least-significant-bit (**LSB**).

In regard to bitwise operators, and within the context of embedded systems programming, one can manipulate the values of each register by either setting or clearing bits. Setting a bit means that a specific bit value in a register will be changed to a **1** without altering the values of the other bits, regardless of the current value of the register. For example, if BIT4 is to be set in a register, then this can be accomplished by doing the following: register = register | BIT4.

To clear a bit means that a specific bit value in a register will be changed to a **0** without altering the values of the other bits, regardless of the current value of the register. For example, if BIT4

is to be cleared in a register, you will first have to negate the bit to turn it into a **0** while changing all other bits in the value to **1**, then do a bitwise and operation. To clarify, the binary representation of BIT4 is 0001 0000. Negating BIT4, i.e. ~BIT4, will return 1110 1111. To complete the explanation, the following is done to accomplish clearing a bit: register = register & (~BIT4).

Objective: The purpose of this assignment is to provide background into embedded systems programming, obtain some familiarity with bitwise operators, and to get back into a programming shape. All of the required functions are not meant to tricky and are the most straight-forward I have ever created.

Thorough explanations were provided in the concepts of setting and clearing a bit. Please ponder for a moment on how to accomplish a bit extraction (i.e. you want to obtain the value of a specific bit in a register) and how to check for the status of a bit, as described in the **Required Functions** section of the assignment.

Required Functions

bit_status(num, bit)

Description: Determine if the passed *bit* is present in the passed number. In other words

Output: This function should not print anything onto the screen.

Return Value: Return **True** if the specified bit is set. In other words, the value of the bit is 1. Return **False** otherwise.

set_bit(num, bit)

Description: Set the specified *bit* into the passed number *num*. In other words, have the bit value be a 1.

Output: This function should not print anything onto the screen.

Return Value: A value with the set bit.

clear_bit(num, bit)

Description: Clear the specified *bit* from the passed number *num*. In other words, have the bit value be a 0.

Output: This function should not print anything onto the screen.

Return Value: A value with the cleared bit.

extract_bit(num, bit)

Description: Obtain the specified bit value of the passed number *num*.

Output: This function should not print anything onto the screen.

Return Value: A value representing the extracted bit.

Testing

Total tests: 32

Testing will be done using the **bits_and_pieces_part2_tester.py**

NOTE – The *bit_status()* function is used in the testing of the *set_bit()* and *clear_bit()* functions to determine if the bit was successfully set and/or cleared.

Notes

Name your program "bits_and_pieces.py", otherwise the program tester will NOT work.

If you have any questions on this assignment, or encounter an issue or bug that you are not able to isolate and figure out on your own AFTER consulting with your peers, please email me at diazg.gustavoa@knights.ucf.edu.