Number Systems and Boolean Logic

Learning Outcomes:

After completing the exercises in this lab you should be able:

- Use the Python language to exercise low-level bitwise operations on integer quantities and demonstrate the fundamental building blocks of modern computer architectures

Organisation

Please attempt this lab individually as you will need this to be completed in order to complete subsequent labs.

1. Start up your Python environment

This lab will involve some Python programming so fire up the Python environment of your choice. Problems marked with an asterisk (*) are optional challenge problems

2. Converting between number systems

In class you were introduced to various manual techniques for converting between decimal, binary and hexadecimal number systems. The Python language provides built-in functions to perform such functions for you.

2.1 Using the bin() and hex() functions, write Python code which lists the first 16 decimal numbers starting from 0 alongside the binary and hexadecimal values for each

2.2 Using modulo arithmetic write an implementation of the hex() function called myhex() which takes a decimal number input and converts it to its hexadecimal representation as a string

3. Boolean identities

In class you saw how the function

F = ~XYZ + ~XY~Z + XZ

could be reduced to a simpler form using the Boolean identities

3.1 Write Python code to verify both of the DeMorgan's laws

3.2 Write a Python function which implements the function F above

3.3 Write a Python function to generate a truth table for a specified number of variables which you can pass as an array of strings

***3.4**. Extend the function from 3.2 to pass in a second parameter, a expressed as lambda function, which operates on the generated truth table values to produce the output function in the rightmost column of the table

4. Cryptography

The XOR operation, can be used to implement a simple but effective cipher.

4.1 Write an encryption function which takes as its inputs two strings representing the plaintext message M and the secret key K and returns as a string, the ciphertext C

4.2. Write an decryption function which takes as its inputs two strings representing the ciphertext C and the secret key K and returns as a string, the plaintext M

4.3 For chosen plaintexts and keys, observe how, if the a key is used to encrypt the same message more than once, that the ciphertext output is the same. Although an attacker can only see ciphertext, if that ciphertext appears to repeat then he might conclude that the key is being reused and be able to start guessing plaintexts.

***4.4** Write a function to generate random keys for use as encryption keys for the XOR cipher.

5. Bit processing

The Python language supports a number of bitwise operators as follows:

&	AND
|	OR
^	XOR
~	NOT (One's complement)
<<	Left shift
>>	Right shift

These can be composed in various ways to operate directly on binary numbers. Using Python, implement the following

5.1 A function called **`pow2()`** which takes an integer input and returns 2 raised to that power

5.2 A function which takes a binary input (represented as a string) and returns its decimal equivalent. There are a couple of ways this can be solved but try using the OR operator to create the return value

5.3 A function which converts a decimal number into a binary number not using the modulo arithmetic technique demonstrated in class. **Hint**: Think about how AND and bit-shifting might be used

5.4 A function which takes as its inputs:

- An integer target

- A start index in the range 0 to 30
- An end index in the range 1 to 31

and returns the binary value of the of the target between the start and the end bit offsets. For example, in the following (8-bit) binary input

0	**1**	**1**	**0**	**1**	**1**	**0**	**1**

the binary value when start=2 and end=6 is 11011. Remember the indexing starts from the lowest significant bit (the rightmost)

Verify your answers with test inputs and the bin() function.

5.5 A function which rotates a 32 bit number left or right by a specified number of positions. Rotation is like shifting but instead of bits dropping off the end of the left- or right-hand side, the bit is inserted back in on the opposite side

6. Adders

Implement binary addition using only bitwise operations in Python

6.1 Implement a half adder which takes two single bit inputs and returns the sum and carry

6.2 Implement a full adder which takes two single bit inputs and a carry-in and returns the sum and carry-out

6.3 Implement a function which adds two 4-bit binary numbers using the full-adder from task 6.2