CAB203
Tutorial 2

Bits

1. Count:
   (a) How many different 8-bit strings are there?
   (b) How many gigabits are there if you have one bit for every possible 32-bit string?
   (c) How many bits are required if you need 18 unique bit strings?
2. Calculate the following expressions using bitwise operations:
   (a) $0011 \,\&\, 1010$
   (b) $0011 \,|\, 1010$
   (c) 0011 ˆ 1010
   (d) ~ 1010
3. In the following, explain how to use bitwise operators to manipulate individual bits.
   (a) In $\overline{x} = 1010$, turn on bit 2
   (b) In $\overline{x} = 1100$, turn off all but bits 1 and 2
   (c) In $\overline{x} = 1110$, flip bit 3
   (d) In $\overline{x} = 0110$, turn off bit 1

Python bit manipulation

1. Define a Python function that takes two integers, x and j, and returns a value which is x, but with bit j set (set to 1).
2. Define a Python function that takes two integers, x and j, and returns a value which is x, but with bit j cleared (set to 0).
3. Define a Python function that takes two integers, x and j, and returns a value which is x, but with bit j flipped.
4. Define a Python function that takes two integers, x and j, and returns True if the jth bit of x is 1, otherwise False.

Character and text representations

1. Find the ASCII representations of the following using the ASCII chart from the Lecture 2 slides
   (a) “A”
   (b) “1”
   (c) “;”

2. Give the C-string for "CAB"
3. Put these string in lexicographic order: 1010, 11, 00110, 110011, 00001
4. *(Stretch question)* The modern system for character encodings is *Unicode.* Unicode is actually a family of encodings for a common set of characters. The most common encoding is UTF-8, which is a *variable length encoding.* Some characters require only 8 bits to encode, and others require 16, 24 or 32 bits. How do you think this is accomplished? In particular, when looking at, say, 8 bytes. How do you know where the characters start and end if they can have different lengths?

Integer representations

1. Convert these binary numbers to base-10
   (a) 111
   (b) 1010
2. Add these binary numbers
   (a) 101 + 011
   (b) 1110 + 1010
3. Convert these binary strings to hexidecimal
   (a) 11001011
   (b) 00001001
4. Convert these hexidecimal strings to binary
   (a) AE
   (b) 10
5. Assuming 4-bit numbers with 2's complement encoding, decode the following:
   (a) 1100
   (b) 0101
6. *(Stretch question)* To negate a number x in 2's complement, you need to find the binary representation for $2^n - x$. There is a shortcut to doing this, which is used internally in CPUs. Supposing that $\overline{z}$ is the binary representation for x,
   (a) NOT each bit in $\overline{z}$
   (b) add 1 to the result.

   Show that this procedure produces the binary representation for $2^n - x$. *Hint: look at a binary representation* $x = \sum_{j=0}^{n-1} z_j 2^j$ *and the version after applying the procedure:* $1 + \sum_{j=0}^{n-1}(1 - z_j)2^j$. *And yes, for a bit* b, $\sim b$ *is the same as* $1 - b$.