

Assignment 1: Digital Logic and Representations

Due Sep 29, 2023 @ 23:59

16 points

Task 1: Digital Logic (4 points)

- You are given two 8-bit full adders. Devise a circuit that uses the adders to perform addition of two 16-bit numbers. You may, if necessary, use any additional NAND or NOR gates. Simply describe the motivation of your design. (1pt)
- What is a decoder chip and what is it used for? (1pt) If a decoder has 3 input pins, how many output pins will it have? (1pt)
- A three-bit binary counter receives the following sequence of inputs: 1100101001. What would be the output of the counter at each time step and for each output line? (1pt)

Task 2: Data Representation (3 points)

- Use mathematical induction to prove that k bits can represent 2^k distinct values. (1pt)
- A four-bit architecture uses a two's complement representation system to interpret a 4-bit byte. What are the maximum and minimum values that can be represented in this 4-bit memory? (1pt) What would the answer be if the byte is interpreted as an unsigned integer? (1pt)

Task 3: Integer Representations (4 points)

In this task we will examine the differences between Little-/Big- Endian encodings, as well as sign-magnitude and two's complement representations.

- Write a C program that allocates a byte array in memory. (Note: at least 8 bytes) (1pt)
- Interpret the array pointer as an integer pointer and store the integer 0x04030201 in memory. Print out the bytes in order and determine if the integer was stored in little- or big-endian order. (1pt)
- Write a function that determines automatically if the architecture uses sign-magnitude or two's complement integer representation. Assume you know the endianness of the system. Hint: choose an appropriate integer to represent in memory as a byte array and compare the byte string to what you would expect to see under each representation. (2pt)

Task 4: Floating-point representation (5 points)

In this task we will emulate an 8-bit floating point representation. We will assume a representation with a one-bit sign, 4-bit normalized mantissa and 3-bit biased exponent.

- a. What are the smallest/largest numbers you can represent? (0.5pt) What is the smallest increment (i.e., step change between two neighboring numbers)? (0.5pt)
- b. Write a function `char toFloat(float arg)` that takes a standard float variable, converts and stores it in a single byte and returns it as a `char` variable. (2pt)
There are multiple possible ways to solve this problem. The recommended way is to use the function `log2f` to calculate the exponent. Next, check for limits and add the bias. Finally, copy the parts of the mantissa corresponding to the most significant bits. Make sure to shift and convert variables as appropriate.
- c. Write a function to print out the converted char variable to screen as a bit string. Use the function to verify that your encoding works correctly for a range of input numbers. (2pt)