Datorteknik
Lab 1: Digital Logic Circuits

Todor Stoyanov

August 4, 2020

1 Objectives and Lab Materials

In this lab you will examine different integer and floating point representation formats.

The matrerials for this lab are:

- Any computer with ThinLinc client installed.
- A ThinLinc server running Ubuntu.

Complete the following tasks. Provide all source code, including appropriate inline comments. In the report explain how the code works and describe what test you performed the validity of your code.

1.1 Preliminaries

Before starting with this lab please verify you have performed these preliminary steps:

- Start a ThinLinc client. Connect to thinlinc.oru.se (or if it doesn't work try 130.243.110.30), using your EDUNET credentials.
- You are now running on one of three ThinLinc server nodes, running Ubuntu Linux. Note: to get out, simply log out from the top right corner.

2 Task 1: Integer Representations (10 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)

- 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.

- 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.

3 Task 2: Floating-point representation (15 points)

In this task we will emulate an 8-bit floating point representation. Complete the following steps:

- Define a new C struct that contains a private data member of type `char`. This will be the data buffer we use to store our floating-point representation.

- We will assume a representation with a one bit sign, 4-bit mantissa and 3-bit exponent. What are the smallest/largest numbers you can represent? What is the smallest increment?

- Write a member function that takes a standard `float` variable, converts and stores it in your representation. As we are simplifying here, you are allowed to use any standard C float and int operations (divisions, modulo, typecasting). Start by enforcing the limits you determined in the previous point. Convert the whole part of the number to a short int and write it as a binary number into a character array (e.g., using `snprintf`). Convert the fractional part to a binary fraction and write it into the same array, while keeping track of where the binary dot should be. Convert the mantissa to cannonical form and calculate the value of the exponent. Using bit shifting and bitwise OR/AND operations, write the corresponding values from the character array into the data member.

- Write a function to print out the data value to screen as a bit string. Use the function to verify that your encoding works correctly for a range of input numbers (at least three different non-trivial values, plus at least one edge case).