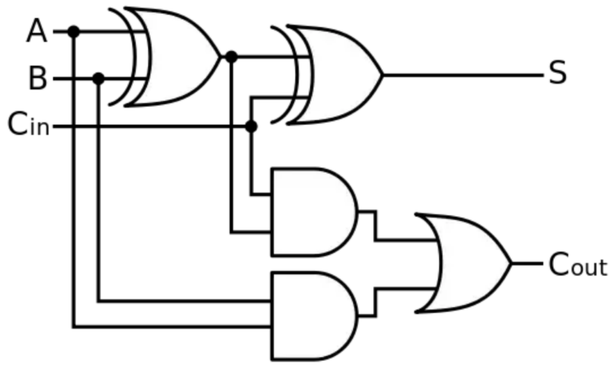
CSCI 2500 — Computer Organization
Lab 05 (document version 1.0)
C the logic in C

- This lab is due by the end of your lab session on Wednesday, October 27, 2021.
- This lab is to be completed **individually**. Do not share your code with anyone else.
- You **must** show your code and your solutions to a TA or mentor and answer their questions to receive credit for each checkpoint.
- Labs are available on Mondays before your lab session. Plan to start each lab early and ask questions during office hours, in the Discussion Forum on Submitty, and during your lab session.

We're back to our ultra high level C programming. This might possibly be a relief after all of the MIPS we've been doing. This lab is going to involve implementing simulations for logic gates and more complex circuits. You can do this through the boolean logic operators available in C (e.g., `||`, `&&`, `^`, etc.). Download the `lab05.c` template to get started. This has empty functions for all of the checkpoints and instructions on what to implement. You'll be filling in all of the listed functions.

1. **Checkpoint 1:** For the first checkpoint, you'll implement 2-input logical AND, OR, and XOR gates. Logical NOT is already implemented for you as an example. From the downloaded template, fill in the provided functions and run the unit tests to verify your outputs for the truth tables are correct.

2. **Checkpoint 2:** Using the gates you constructed in Checkpoint 1, you'll next be implementing a 2-input decoder, a 2-input multiplexor, a 4-input multiplexor, and a 1-bit adder. See the corresponding slides in the Chapter-3a slide deck for the logic designs of the decoder and multiplexors. See the truth table for the 1-bit adder, and see below for one of multiple possible implementations. The intention is to use your implemented gates and circuits to construct more complex circuits. E.g., you can use the 2-input decoder in the construction of your 4-input multiplexor.



3. **Checkpoint 3:** Finally, you'll use all of the above to implement the 1-bit ALU given in the Chapter-3a slide deck. Note that *Binvert* is an input for a 2-input multiplexor, and *Operation* is a 2-bit input to a 3-input multiplexor – for this, you can just use the 4-input mux you designed and set the fourth input to `FALSE` to ignore it. Also, if both Operation bits (`Op0` and `Op1`) are set to `TRUE`, then you'll set the `Result` bit to be `UNDEF`.