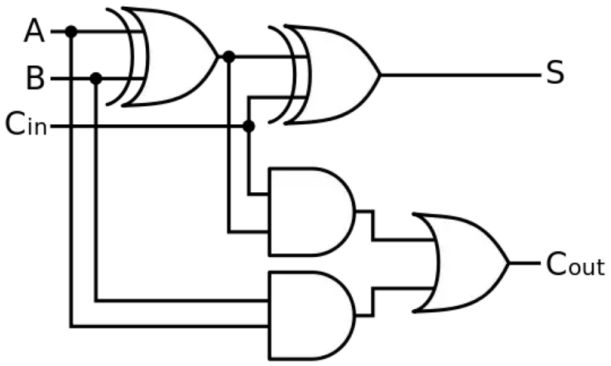
CSCI 2500 — Computer Organization
Lab 05 (document version 1.0) — Due 22 March 2023
C the logic in C

- This lab is will be graded in your Wednesday lab session.
- This lab is to be completed **individually**. Do not share your code with anyone else.
- Labs are available on Mondays before your lab session. Plan to start each lab early and ask questions during office hours, in the Submitty forum, 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 and x86 we've been doing. This lab is going to involve implementing simulations for logic gates and more complex circuits. You should do this with **only** the boolean logic operators available in C (`!`, `||`, `&&`). Download the `lab05.c` template to get started. You will be filling in all of the listed functions.

1. **Checkpoint 1:** For the first checkpoint, you will implement 2-input logical AND, OR, XOR, and NOR 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. You will need to implement NOR and XOR using AND/OR/NOT gates, as we've seen in class.
2. **Checkpoint 2:** Using the gates you constructed in Checkpoint 1, you will next be implementing a 2-input decoder, a 2-input multiplexor, a 4-input multiplexor, and a 1-bit adder. See slides 7, 20, and 22 in the Chapter-3b slide deck for the logic designs of the decoder and multiplexors. See the truth table on slide 14 in Chapter-3d 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 on slide 25 of the Chapter-3d 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 4th 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 `FALSE`. You can easily use a 2-mux with the selector bit set to `and_gate(Op0, Op1)`, one input as the selected output from the ALU, and the other input as `FALSE` to accomplish this. **Note how this is effectively representing if-else logic.**

To get full credit, you will need to validate your outputs against the Submitty gradeable. However, **you will still need to be checked off for the lab** from the mentors and TAs to get credit. The gradeable is for validation only.

Assignment Rules

We don't usually have much in the way of explicit rules for our labs and assignments, but these logic gates simulations will be a special case.

The primary purpose of this lab is to demonstrate how basic logic gates can be combined together into complex circuits. As a rule, whatever you implement in your solution file should be a loose representation of an implementable circuit. This is the only difference between our C implementations and what we'd be doing in Verilog, as Verilog would inherently be enforcing these *rules*.

The below are guidelines to this end. These rules must be followed **except** where explicitly noted in the assignment documents or template code: :

(a) No modifications to main().

(b) You are allowed to implement additional basic gates – e.g., a three input AND gate – to simplify your other circuits.

(c) If desired, you can modify the function prototypes given in the template. For example, you can pass an additional control bit to your ALU for the sake of outputting NOR instead of OR.

(d) No if() or if()-else() logical control statements, or equivalent.

(e) You can use basic for loops only – `for (int i = 0; i < N; ++i)`. Different loop structures are not allowed, as they can easily be used as a logical equivalent to if-else statements.

(f) No calls to any external or library functions.

(g) The above rules are to basically ensure that you're sticking to using gates for all logical control and operations. No tricky workarounds are allowed.

Note: Most of these rules are going to be much more relevant to HW5 and the Project.