Sequential Logic Circuits; SDC Framework

CS 350: Computer Organization & Assembler Language Programming
Lab 7, due Fri Mar 28

A. Why?

- Storage elements are the basic circuits that store data, which is used in logic circuits that use memory.
- Finite state machines (FSMs) use sequential logic circuits and can model simple programs with fixed memory.
- Implementing the von Neumann architecture helps you understand how it works.

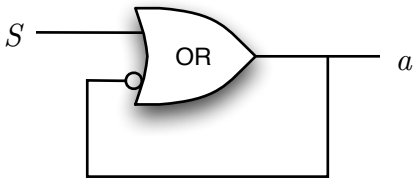
B. Outcomes

After this lab, you should be able to

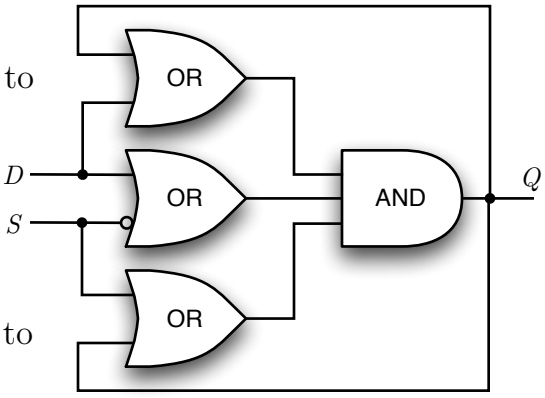
- Be able to determine whether a logic circuit has a logically stable state.
- Design simple FSMs to recognize a simple pattern of characters or generate a simple pattern of output.
- Translate between FSMs state transition diagrams and tables and use them to trace FSM execution.
- Code (in C) the framework of a simulator for a simple von Neumann computer.

C. Written Problems [50 points total]

1. [12 = 3 * 4 pts] Consider the circuit to the right. (a) Describe the new value of a as a function of S and the current value of a. (b) When (if ever) does this circuit have a logically stable values for a? A logically unstable value for a? (c) Can this circuit be used to remember a bit?



2. [$12 = 3 * 4$ pts] Consider the logic circuit to the right. (a) Describe the new value of Q as a function of D, S, and the current value of Q. (b) When (if ever) does this circuit have logically stable or unstable values for Q? (c) Can this circuit be used to store a bit? (I.e., does it remember a bit? Can we set the bit when we wish?)



3. [4 pts] Say we have a calculator with no extra memory, just a 32-bit number being displayed. What are the address size and addressability of this machine?

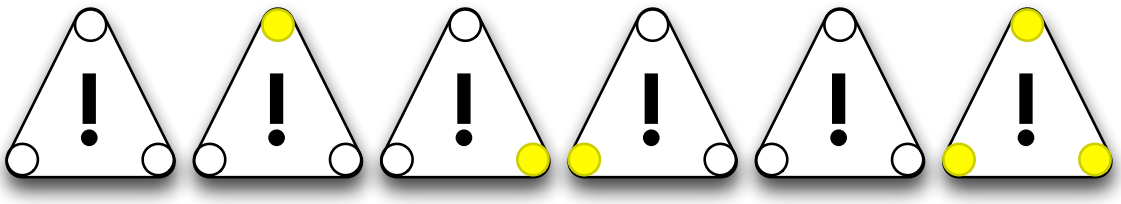
4. [4 pts] What are the address size and addressability of a programmable calculator with 4096 memory locations where each location can hold 24 bits?

5. [18 points] Design a finite state machine that reads strings of `0`'s and 1's and accepts iff the input contains a `1011`. (The `1011` may appear anywhere inside the input; it can even be the whole input.) Give both state transition tables and diagrams and show a trace of execution on the input `01010110`, which should be accepted. (When giving the state transition table version of the machine, don't forget to also state what the start state is and what the accepting state(s) is/are.)

6. [20 points] Design a finite state machine that reads strings of `a`'s, `b`'s, and `c`'s and accepts iff each letter occurs an even number of times. For example, `aaaa` and `abacbc` should be accepted but `aaa` shouldn't (the third `a` needs a mate); nor should `cabac` (the `b` is missing a mate). Note: 0 is an even number, so the empty string is an acceptable input.

   As for the previous problem, give state transition table and diagram representations of your machine. Show traces of execution on the inputs `abacbc` and `cabac`. (Hint: you need at least 8 states.)

7. [30 points] Let's modify the flashing arrow example from the book so that we have a triangular sign with one light in each corner; clockwise from the top, let's call them lights 1, 2, and 3. When the switch is on, the lights flash in the order None, #1, #2, #3, None, All three, … (and repeat).



Give a finite state machine description for this sign (show both a state transition table and a state transition diagram), and include three bits of output with each state (one bit each for lights 1, 2, and 3). Also, give logic expressions that implement the combinatorial part of this machine. You can present them in full DNF form if you want (you don't have to optimize them). In any case, you don't have to draw them out as logic circuits.

