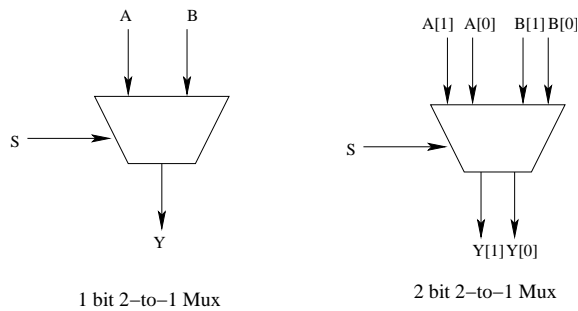


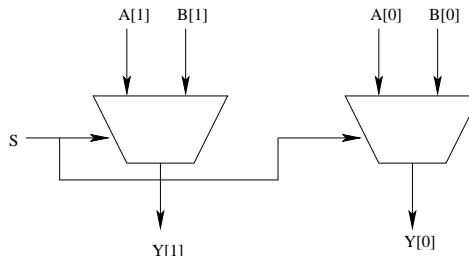
Write your answers on these pages. Additional pages may be attached (with staple) if necessary. Please ensure that your answers are legible. Please show your work. Due at the *beginning of class*. Total points: 34.

1. [6 Points] **Muxes and Memory.**

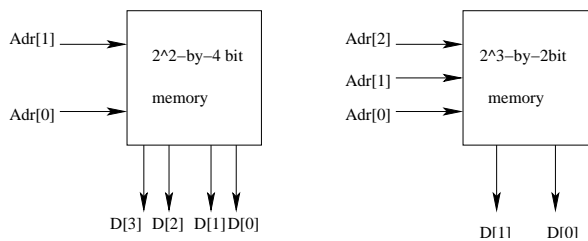
- (a) A 2-bit 2-to-1 mux (below, right) is similar to a 1-bit 2-to-1 mux (below, left) except the former selects among 2-bit inputs rather than 1-bit inputs. More specifically, the output of 2 bit 2-to-1 mux is defined as follows: if $S = 0$, $Y_i = A_i$ and if $S = 1$, $Y_i = B_i$, for $i = 0, 1$. Construct a circuit with the behavior of a 2-bit 2-to-1 mux using two 1-bit 2-to-1 muxes.



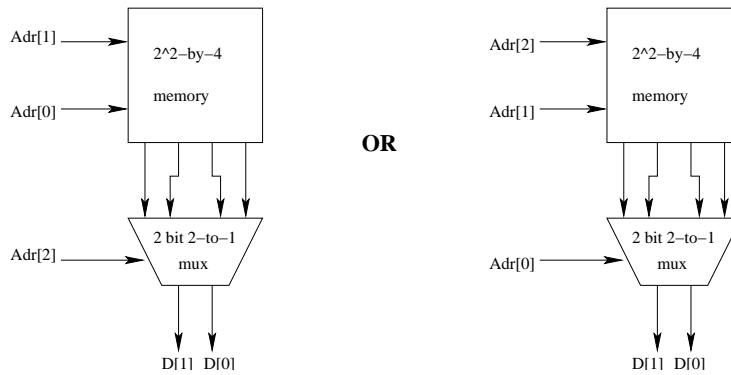
Answer:



- (b) Memories come in many shapes and sizes. If we need a memory of a particular size and addressability, we can often construct it out of other, differently-structured memories. Construct a 2^3 -by-2-bit memory (below, right) using a 2 bit 2-to-1 mux and a 2^2 -by-4-bit memory (below, left). Assume that the memories are read-only memories (ROMs), so you need only be concerned with reading.



Answer:



2. [8 Points] **Combinational Logic Circuits.** Do you remember the game of tic-tac-toe? (See p. 73–74 in your textbook if you do not.) We can use bit vectors to represent the state of any tic-tac-toe game. Consider the following game state (we use “*” instead of “O” to avoid confusion with zero).

X	*	X
	*	

We can represent this state using two 9-bit vectors $\mathbf{AX} = (101\ 000\ 000)$ and $\mathbf{A}^* = (010\ 010\ 000)$. The vector \mathbf{AX} indicates whether each cell contains an **X** (denoted with a 1) or a non-**X** (denoted with a 0). The first three bits represent the top three cells from left to right; the next three bits represent the middle three cells; and the last three bits in the vector represent the bottom three cells. Vector \mathbf{A}^* is analogously defined.

- (a) Give the bit-vector representation (both \mathbf{AX} and \mathbf{A}^*) for the following game state.

X	*	X
	*	*
	X	X

Answer:

$\mathbf{AX} : (101\ 000\ 011)$

$\mathbf{A}^* : (010\ 011\ 000)$

- (b) Give the bit-vector representations (both \mathbf{AX} and \mathbf{A}^*) of all possible next states for the above game, assuming it is *’s turn next.

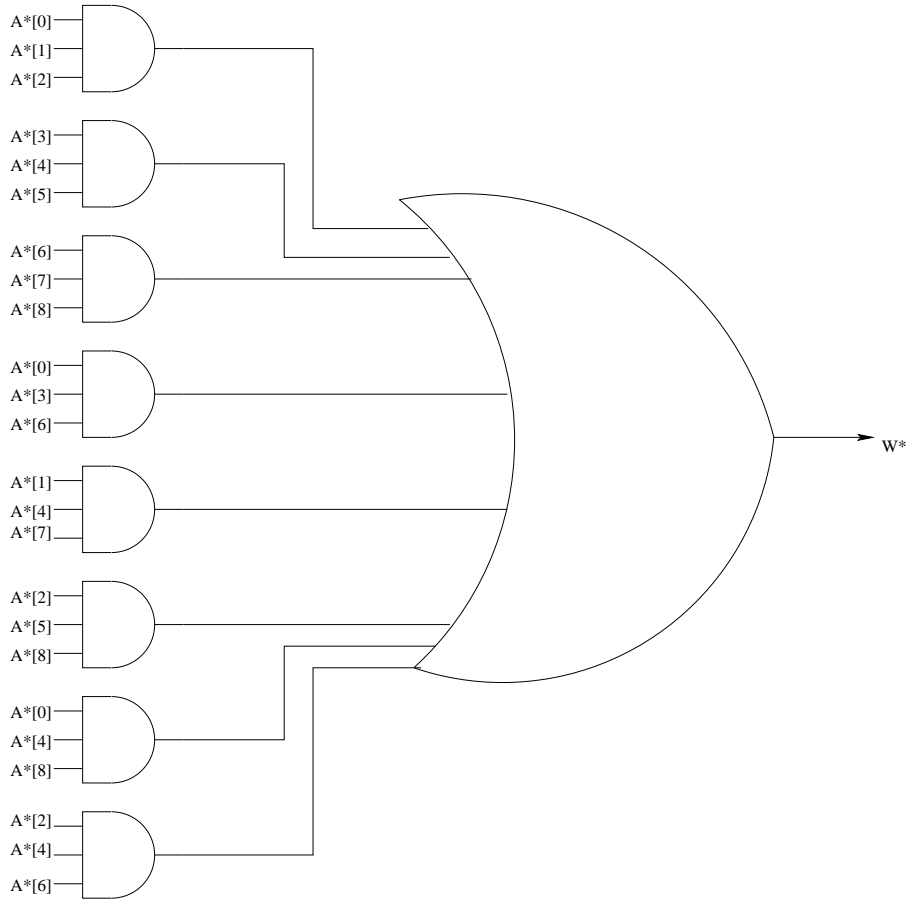
Answer:

i. $\mathbf{AX} : (101\ 000\ 011)$. $\mathbf{A}^* : (010\ 111\ 000)$

ii. $\mathbf{AX} : (101\ 000\ 011)$. $\mathbf{A}^* : (010\ 011\ 100)$

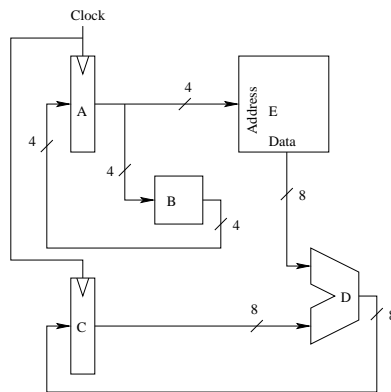
- (c) Construct a gate-level logic circuit to determine if the * player has won (*i.e.*, has three in a row). The input to this circuit is the 9-bit A^* vector ($A^* [0-8]$). The output (W^*) should be 1 if only if the * player has won. You may assume the game state was arrived at through a succession of legal moves (*i.e.*, both players will not have 3 in a row).

Answer:



There are eight ways in which * can win. *i.e.* by placing 3 consecutive * 's along the 3 rows, the 3 columns, or the 2 diagonals. Each of these eight cases can be implemented using a 3 input AND gate which checks if the 3 appropriate cells are marked by a * or not. The final answer is an OR of all these eight cases.

3. [8 Points] **Sequential Logic Circuits.** Consider the following circuit.



Components are labeled according to the following key: (A) A 4-bit register representing an unsigned binary number (initially set to 0). (B) A 4-bit incrementer. (C) An 8-bit register (initially set to 0). (D) An 8-bit adder. (E) A 2^4 -by-8-bit memory. Assume the memory has the following contents.

Address	Contents	Address	Contents
0	2	8	5
1	1	9	2
2	1	10	1
3	4	11	1
4	2	12	3
5	3	13	4
6	2	14	1
7	2	15	1

- (a) Determine the values stored in the A and C registers at the end of each clock cycle for 8 cycles. Complete the following table with this information.

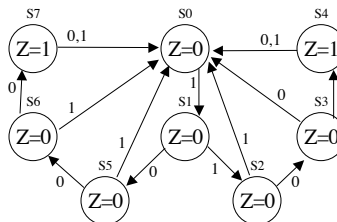
Clock Cycle	A	C
0	0	0
1	1	2
2	2	3
3	3	4
4	4	8
5	5	10
6	6	13
7	7	15
8	8	17

- (b) In a sentence, what does this circuit do?

Answer: This circuit sums the contents of memory starting at address 0.

4. [12 Points] **Finite State Machines.** It's your first day at the Pretty Good Lock Co. Your boss knows you're a whiz so she asks you to build a combination lock circuit. This lock takes a combination (*i.e.*, a sequence of 0s and 1s on the 1-bit input X) and generates a 1-bit output Z, indicating whether the lock is closed (0) or open (1). The lock must accept two combinations: 1, 1, 0, 1 and 1, 0, 0, 0.

- (a) Where do you begin? First, you examine the state diagram (below) created by your predecessor (he got fired; don't ask why). Assume S0 is the initial state. Edges are labeled with the input that causes a state transition. Nodes are labeled with the output (*e.g.*, "Z=1") of that state.

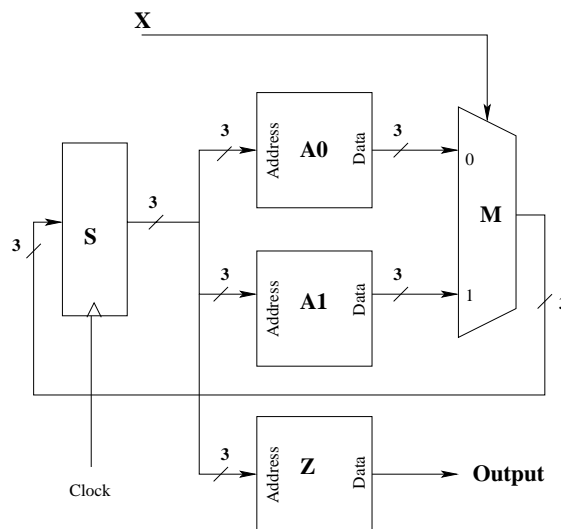


Experiment with the finite state machine defined by this state diagram to determine if it actually accepts 1, 1, 0, 1 and not, for example, 1, 1, 0, 1, 0 by completing the following table. (S-next represents the state reached when a transition is made from state S on input X. Z is the output in state S-next.)

Answer:

S	X	S-next	Z
0	1	1	0
1	1	2	0
2	0	3	0
3	1	4	1
4	0	0	0

- (b) So far so good. You start to build a circuit for the state diagram, but you suddenly realize that once you build the circuit, you will be unable to change the combination without building a new circuit. Maybe that's why your predecessor got fired! You resolve to do better and build the following programmable combination lock circuit.



Components are labeled according to the following key: (A0) A 2^3 -by-3-bit memory. (A1) A 2^3 -by-3-bit memory. (Z) A 2^3 -by-1-bit memory. (S) A 3-bit register (initially containing 0). (M) A 3-bit 2-to-1 mux. Suppose the contents of memories A0, A1 and Z are initialized as follows.

Address	A0	A1	Z
0	1	0	0
1	0	2	0
2	3	0	0
3	0	4	0
4	5	0	0
5	0	6	0
6	0	0	1
7	0	0	0

Voila! You've built a programmable combination lock, and it's currently programmed to accept the combination 0, 1, 0, 1, 0, 1. Try it out on 0, 1, 0, 1, 0, 1, 1 by completing the following table.

Answer:

Clock Cycle	S	X	S-next	Z
0	0	0	1	0
1	1	1	2	0
2	2	0	3	0
3	3	1	4	0
4	4	0	5	0
5	5	1	6	1
6	6	1	0	0

- (c) Okay, cool. Program this circuit (*i.e.*, provide the contents of the memories) to accept the two combinations from part (a). Complete the following table.

Answer:

Address	A0	A1	Z
0	0	1	0
1	5	2	0
2	3	0	0
3	0	4	0
4	0	0	1
5	6	0	0
6	7	0	0
7	0	0	1

- (d) In fact, the above circuit can be used to implement any finite state machine with 1 input, 1 output, and up to 8 states. (i) How would you modify your circuit to support more than 8 states? (ii) How would you modify your circuit to support 2-bit inputs? Don't actually build the circuit; instead, describe what changes you would make to the existing circuit.

Answer:

- i. We use our states as addresses in our A0, A1, and Z memories (*i.e.*, based on the state we use the memories to determine the next state or the output). If we have more states, we need additional address lines, implying we need larger memories. For example, if we have 16 or fewer states, we need 4 address bits and memories with a 2^4 address space. Similarly, each word in the A0 and A1 memories must be 4-bits wide to specify the next state (of 16 possibilities).
- ii. Note that we have a memory (*e.g.*, A0, A1) for each possible input. With only one input line, we can only have two inputs (0 or 1), so we only have two memories, A0 and A1. If we have two input lines, we can have four possible inputs, 00, 01, 10, 11. We'll need a memory for each one as well as a 4-input mux. In general, if we have n input bits, we will need 2^n memories and a 2^n -to-1 mux. The sizes of all memories and the S register remain the same (they depend only on the number of states, not on the number of inputs).