Department of Electrical and Computer Engineering

The University of Texas at Austin

EE 306, Fall 2021
Problem Set 3 Solution
Due: **October 11th**, before class
Yale N. Patt, Instructor
TAs: Sabee Grewal, Ali Fakhrzadehgan, Ying-Wei Wu, Michael Chen, Jason Math, Adeel Rehman

Instructions: You are encouraged to work on the problem set in groups and turn in one problem set for the entire group. **The problem sets are to be submitted on Gradescope**. Only one student should submit the problem set on behalf of the group, but everyone should create a gradescope account and be tagged on the homework.

1. [Wording improved on 10/5] We want to make a state machine for the scoreboard of the Texas vs. Oklahoma Football game. The following information is required to determine the state of the game:
   1. Score: 0 to 99 points for each team
   2. Down: 1, 2, 3, or 4
   3. Yards to gain: 0 to 99
   4. Quarter: 1, 2, 3, 4
   5. Yardline: any number from Home 0 to Home 49, Visitor 0 to Visitor 49, or 50
   6. Possession: Home, Visitor
   7. Time remaining: any number from 0:00 to 15:00, where m:s (minutes, seconds)

   a. What is the minimum number of bits that we need to use to store the state required?
   (100*100)*4*100*4*101*2*901 = 2912032000000.
   2^41 < 2912032000000 < 2^42 so we need 42 bits

   b. Suppose we make a separate logic circuit for each of the seven elements on the scoreboard, how many bits would it then take to store the state of the scoreboard?
   1. 7 x 2 bits
   2. 2 bits
   3. 7 bits

4. 2 bits
5. 7 bits
6. 1 bit
7. 4 bits for minutes 6 bits for seconds

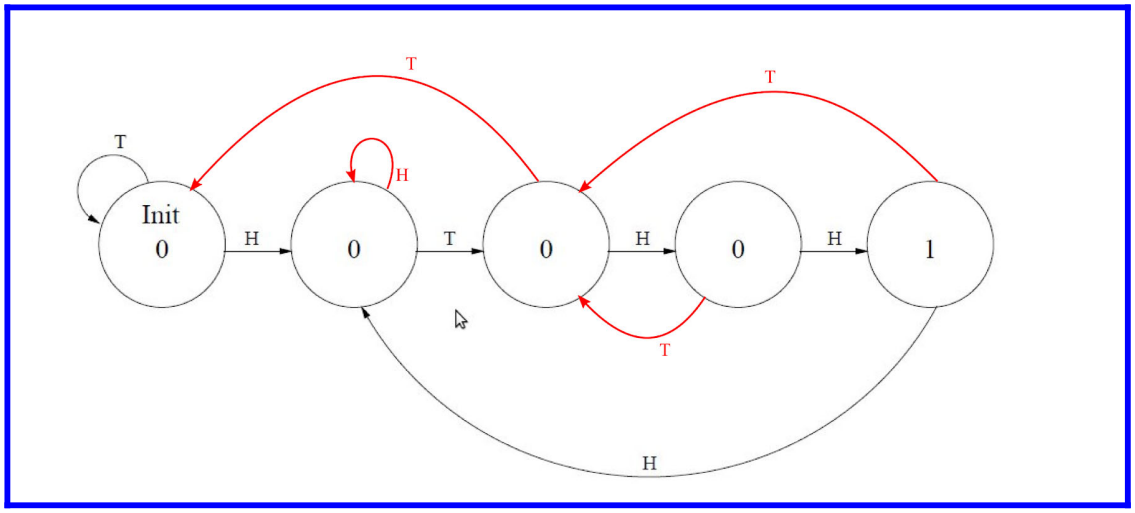
Total 43 bits

c. Why might the method of part b be a better way to specify the state than the method of part a?
The assignments in (b) are easier to decode.

2. [Wording improved on 10/5] Shown below is a partially completed state diagram of a finite state machine that takes an input string of *H* (heads) and *T* (tails) and produces an output of 1 every time the string HTHH occurs.

a. Complete the state diagram of the finite state machine that will do this for any input sequence of any length.

Completed state machine is shown in the figure below



b. If this state machine is implemented with a sequential logic circuit how many state variables will be needed? (Recall, the number of state variables is the same as the number of bits needed to represent all of the states.)

3 bits.

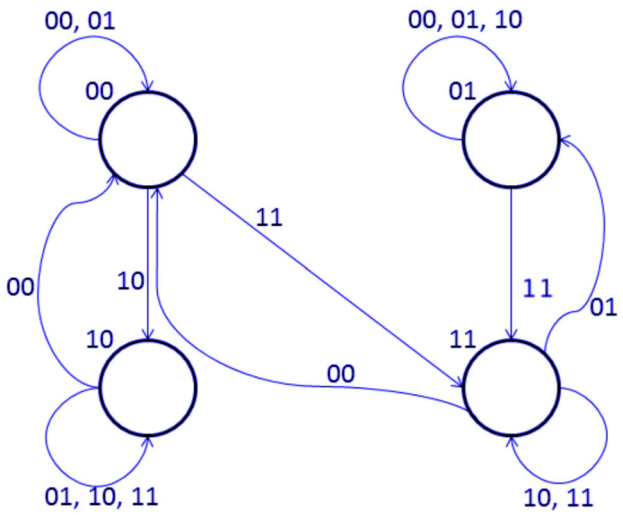
3. (Adopted from 3.37 in the textbook) If a particular computer has 8 byte addressability and a 4 bit address space, how many bytes of memory does that computer have?

Number of byes = address space X adressability. 2^4 x 2^3 = 2^7 = 128 bytes

4. ***Elevator Problem Revisited.***

Recall the elevator controller problem on Problem Set 2. You were asked to design the truth table for an elevator controller such that the option to move up or down by one floor is disabled. If there is a request to move only one floor or to move zero floors, the elevator should remain on the current floor. For this problem, you will design the state machine for the sequential logic circuit for an elevator controller which performs the same operation. You can assume that the building the elevator is in has 4 floors. The input to the state machine is the next requested floor. There will be a state for each floor the elevator could be on. Draw a finite state machine that describes the behavior of the elevator controller. How many bits are needed for the inputs?

Two bits for input. There are technically no output bits, but there are 2 bits needed to represent the current state.



5. (Adopted from 3.39 in the textbook) Using Figure 3.21 on page 78 in the book, the diagram of the, 2^2-by-3-bit memory.
   a. To read from the fourth memory location, what must the values of A[1:0] and WE be?

      To read from the fourth location A[1:0] should be 11, to read from memory the WE bit should be 0. To write to memory the WE bit must be 1.

b. To change the number of locations in the memory from 4 to 60, how many address lines would be needed? What would the addressability of the memory be after this change was made?

To address 60 locations you need 6 bits of address line, which means your MAR is 6 bits . However since we did not change the number of bits stored at each location the addressability is still 3 bits

c. Suppose the width (in bits) of the program counter is the minimum number of bits needed to address all 60 locations in our memory from part (b). How many additional memory locations could be added to this memory without having to alter the width of the program counter?

You need 6 bits for part b, which can address 64 different locations so you could add 4 more locations and not have to increase the width of the program counter.

6. The figure below is a diagram of a 2^2-by-16-bit memory, similar in implementation to the memory of Figure 3.21 in the textbook. Note that in this figure, every memory cell represents **4 bits** of storage instead of **1 bit** of storage. This can be accomplished by using 4 Gated-D Latches for each memory cell instead of using a single Gated-D Latch. The hex digit inside each memory cell represents what that cell is storing prior to this problem.

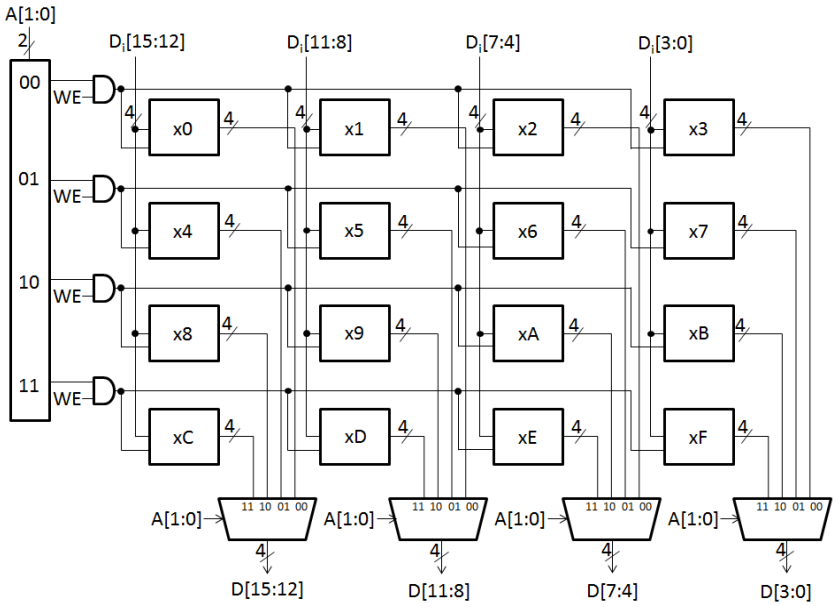


Figure 2. 2^2-by-16 bit memory

a. What is the address space of this memory?

2^2 = 4 memory locations.

b. What is the addressability of this memory?

16 bits.

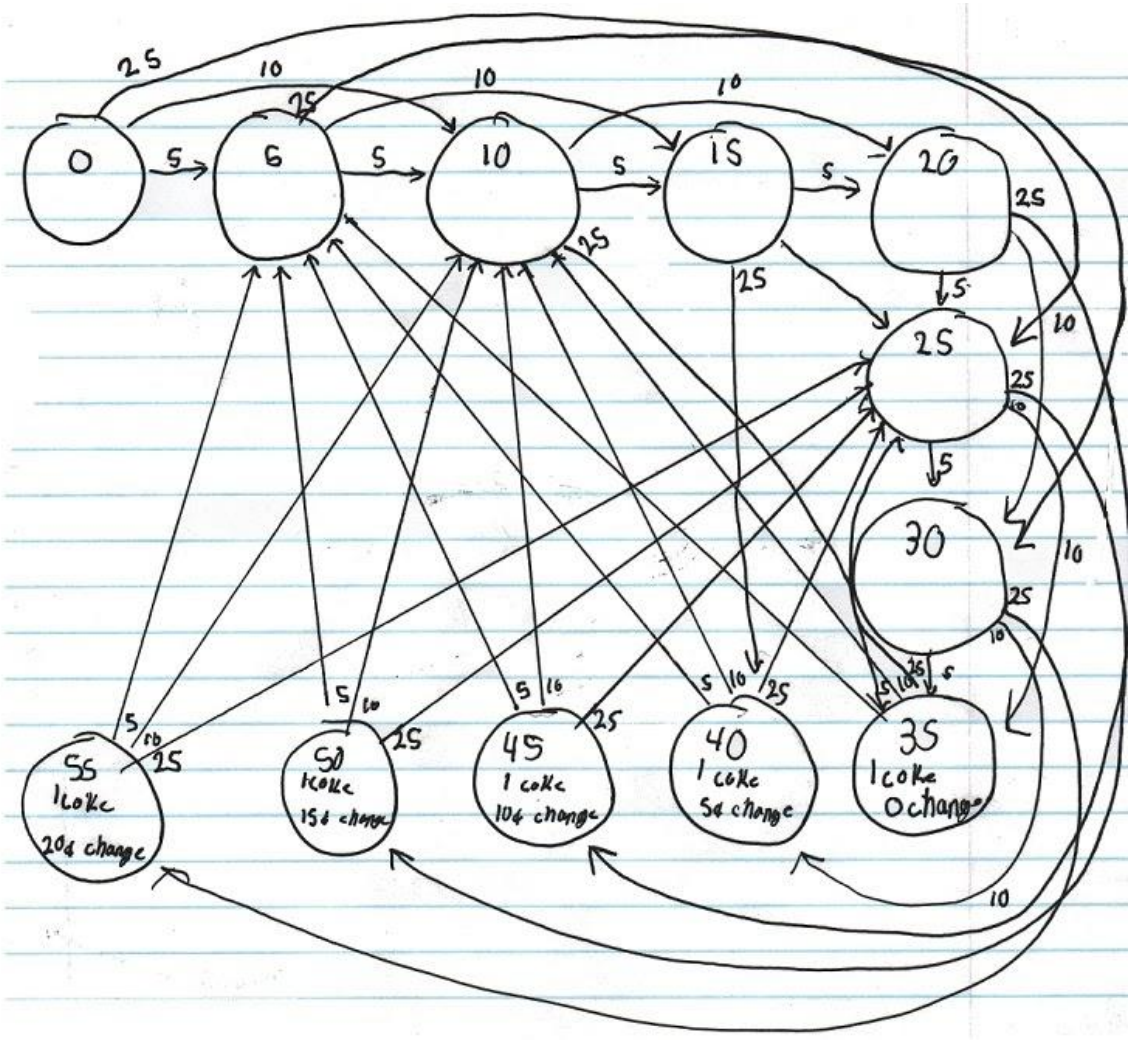
c. What is the total size in bytes of this memory?

8 bytes.

d. This memory is accessed during four consecutive clock cycles. The following table lists the values of some important variables **just before the end of the cycle** for each access. Each row in the table corresponds to a memory access. The read/write column indicates the type of access: whether the access is reading memory or writing to memory. Complete the missing entries in the table.

WE	A[1:0]	D_i[15:0]	D[15:0]	Read/Write
0	01	xFADE	**x4567**	**Read**
1	10	xDEAD	**xDEAD**	**Write**
0	**00**	xBEEF	x0123	Read
1	11	**xFEED**	xFEED	Write

7. (Adopted from 3.47 in the textbook) The Eta Kappa Nu (HKN) office sells sodas for 35 cents. Suppose they install a soda controller that only takes the following three inputs: nickel, dime, and quarter. After you put in each coin, you push a pushbutton to register the coin. If at least 35 cents has been put in the controller, it will output a soda and proper change (if applicable). Draw a finite state machine that describes the behavior of the soda controller. Each state will represent how much money has been put in (*Hint:* There will be seven of those states). Once enough money has been put in it, the controller will go to a final state where the person will receive a soda and proper change (*Hint:* There are five such final states). From the final state, the next coin that is put in will start the process again, contributing to the next purchase.



8. Suppose that an instruction cycle of the LC-3 has just finished and another one is about to begin. The following table describes the values in select LC-3 registers and memory locations.

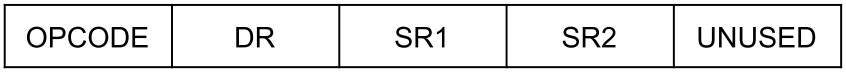
Register	Value
IR	x3001
PC	x3003
R0	x3000
R1	x3000
R2	x3002
R3	x3000
R4	x3000
R5	x3000
R6	x3000
R7	x3000
Memory Location	**Value**
x3000	x62BF
x3001	x3000
x3002	x3001
x3003	x62BE

For each phase of the new instruction cycle, specify the values that PC, IR, MAR, MDR, R1, and R2 will have *at the end* of the phase in the following table.

	PC	IR	MAR	MDR	R0	R1	R2	R3	R4	R5	R6	R7
Fetch	x3004	x62BE	x3003	x62BE	x3000	x3000	x3002	x3000	x3000	x3000	x3000	x3000
Decode	x3004	x62BE	x3003	x62BE	x3000	x3000	x3002	x3000	x3000	x3000	x3000	x3000
Evaluate Address	x3004	x62BE	x3003	x62BE	x3000	x3000	x3002	x3000	x3000	x3000	x3000	x3000
Fetch Operands	x3004	x62BE	x3000	x62BF	x3000	x3000	x3002	x3000	x3000	x3000	x3000	x3000
Execute	x3004	x62BE	x3000	x62BF	x3000	x3000	x3002	x3000	x3000	x3000	x3000	x3000
Store Result	x3004	x62BE	x3000	x62BF	x3000	x62BF	x3002	x3000	x3000	x3000	x3000	x3000

Hint: Notice that values of memory locations x3000, and 3003 can be interpreted as LDR instructions.

9. (Adopted from 4.8 in the textbook) Suppose a 32-bit instruction has the following format:



If there are 255 opcodes and 120 registers, and every register is available as a source or destination for every opcode,

a. What is the minimum number of bits required to represent the OPCODE?

225 opcode, 8 bits are required to represent the OPCODE

b. What is the minimum number of bits required to represent the Destination Register (DR)?

120 registers, 7 bits to represent the DR

c. What is the maximum number of UNUSED bits in the instruction encoding?

3 registers and 1 opcode, 3x7 + 8 = 29 bits. So there are 3 unused bits.

~~**10.** [Removed on 10/6] ***A State Diagram.***~~

~~We wish to invent a two-person game, which we will call "X-and-Y" that can be played on the computer. Your job in this problem is to contribute a piece of the solution.~~

~~The game is played with the computer and a deck of cards. Each card has on it one of four values (X, Y, Z, and N). Each player in turn gets five attempts to accumulate points. We call each attempt a round. After player A finishes his five rounds, it is player B's turn. Play continues until one of the players accumulates 100 points. Your job today is to ONLY design a finite state machine to keep track of the STATE of the current round.~~

Each round starts in the initial state, where X=0 and Y=0. Cards from the deck are turned over one by one. Each card transitions the round from its current state to its next state, until the round terminates, at which point we'll start a new round in the initial state.

The transitions are as follows:

X: The number of X's is incremented, producing a new state for the round.
Y: The number of Y's is incremented, producing a new state for the round.
Z: If the number of X's is less than 2, the number of X's is incremented, producing a new state for the round. If the number of X's is 2, the state of the current round does not change.
N: Other information on the card gives the number of points accumulated. N also terminates the current round.

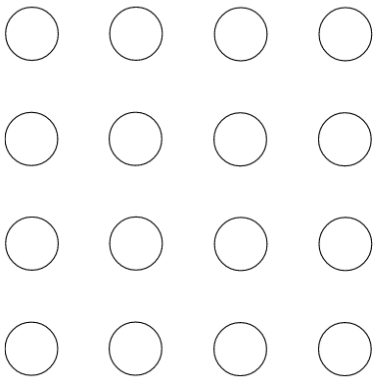
Important rule: If the number of X's or Y's reaches a count of 3, the current round is terminated and another round is started. When a round starts, its state is X=0, Y=0.

Hint: Since the number of X's and Y's specify the state of the current round, how many possible states are needed to describe the state of the current round.

Hint: A state can not have X=3, because then the round would be finished, and we would have started a *new* current round.

On the diagram below, label each state. For each state draw an arrow showing the transition to the next state that would occur for each of the four inputs. (We have provided sixteen states. You will not need all of them. Use only as many as you need).

Note, we did not specify outputs for these states. Therefore, your state machine will not include outputs. It will only include states and transitions represented by inputs.



11. ***Trying Out Flip-Flops.***

The flip-flop we introduced in class is shown below.

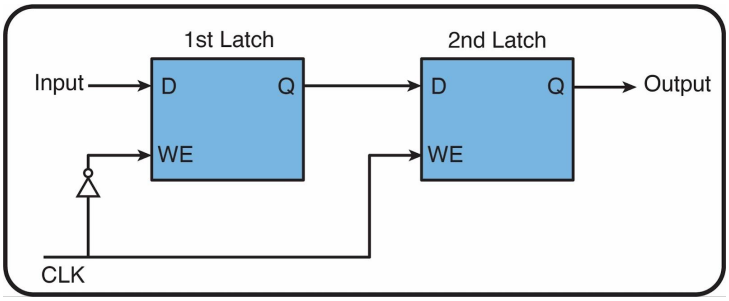
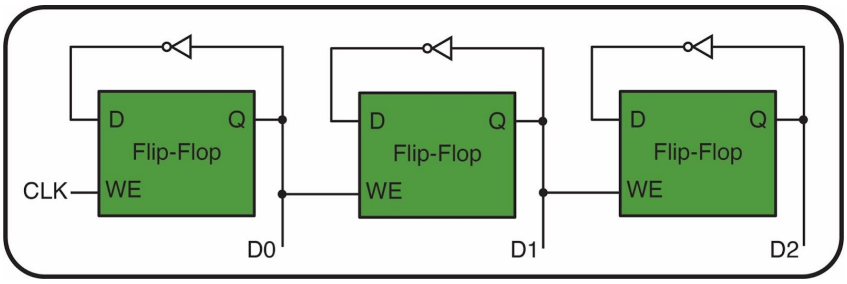


Figure 3. The two-gated-latch flip-flop.

Note that the input value is visible at the output after the clock transitions from 0 to 1. Shown below is a circuit constructed with three of these flip-flops.



Your job: Fill in the entries for D2, D1, D0 for each of clock cycles shown: (In Cycle 0, all three flip-flops hold the value 0).

In 10 words or less, what is this circuit doing?

	cycle 0	cycle 1	cycle 2	cycle 3	cycle 4	cycle 5	cycle 6	cycle 7
D2	0	1	1	1	1	0	0	0
D1	0	1	1	0	0	1	1	0
D0	0	1	0	1	0	1	0	1

inverts on positive "edge" of D1

inverts on positive "edge" of D0

inverts on positive "edge" of clock

"edges" in bold

In 10 words or less, what is this circuit doing?

D2, D1, D0 act as a decrementing counter

12. Write a program in LC-3 machine language that places a 1 into R0 if the value in R1 is identical to the value in R2, and places a 0 into R0 if the values in R1 and R2 are different.

0101000000100000 ; R0 <- 0
1001011010111111 ; R3 <- NOT(R2)
0001011011100001 ; R3 <- R3 + 1; in effect, makes R3 <- -R2
0001001001000011 ; R1 <- R1 + R3; in effect, makes R1 <- R1 - R2
0000101000000001 ; branch to the last instruction if negative or positive
0001000000100001 ; R0 <- R0 + 1 (makes R0 1 instead of 0)
1111000000100101 ; HALT

13. What does the following program do (in 20 words or fewer).

```
0101 100 100 1 00000
1001 000 001  111111
0001 000 000 1 00001
0001 000 000 000 010
0000 100   000000001
0001 100 100 1 00001
1111 0000  0010 0101
```

Makes R4 <- 1 if R2 >= R1; else, makes R4 <- 0.

14. What does the following program do (in 20 words or fewer).

```
0101 000 000 1 00000
0101 101 001 1 00001
0000 101   000000001
0001 000 000 1 00001
1111 0000  0010 0101
```

Makes R0 <- 1 if R1 is even; if R1 is odd, makes R0 <- 0.