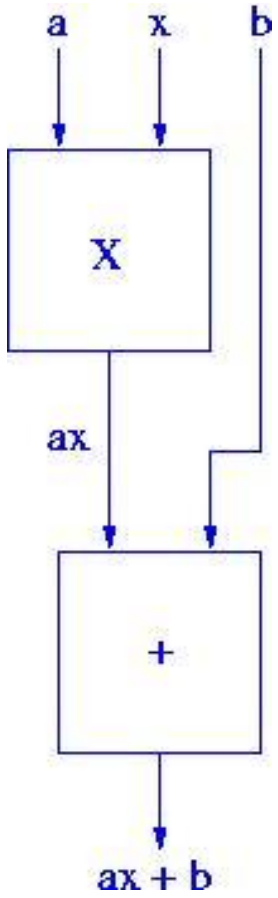
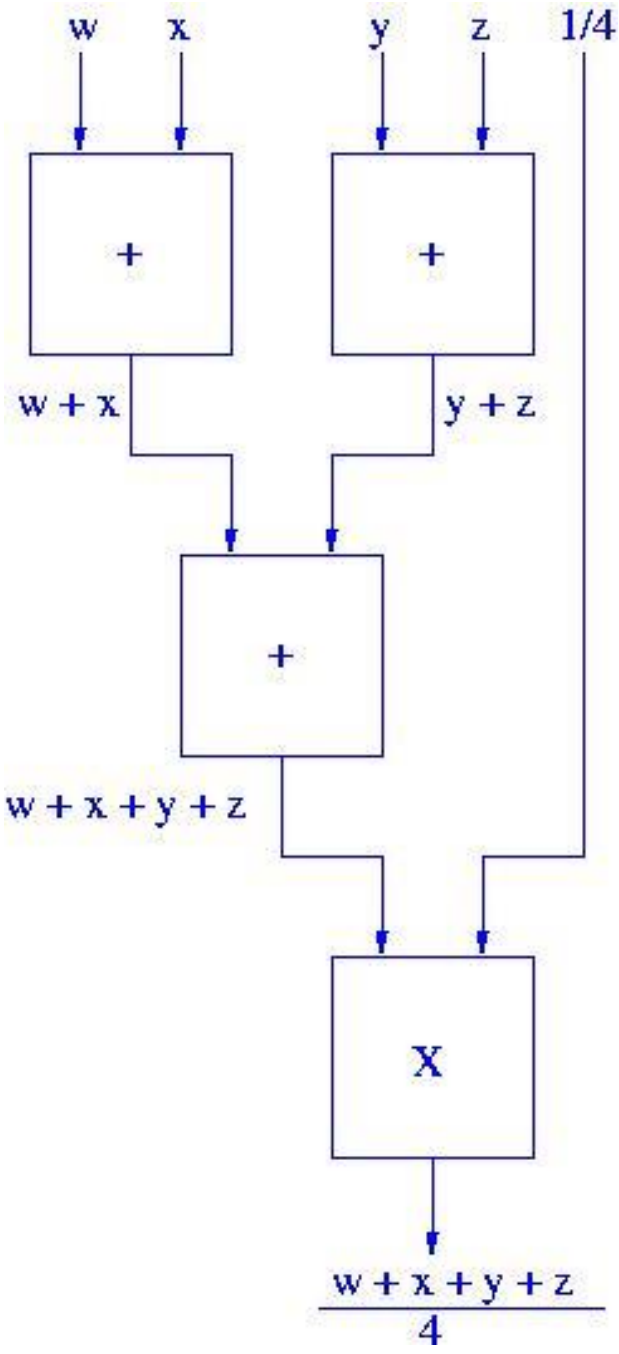
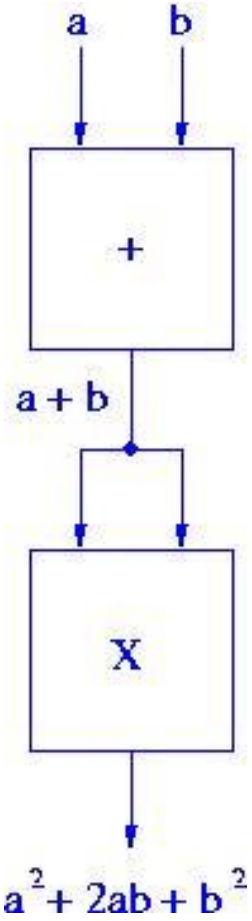
1. (Adapted from problem 1.5 in the textbook)
   Say we had a "black box," which takes two numbers as input and outputs their sum. See Figure 1.10a in the Textbook or the following figure. Say we had another box capable of multiplying two numbers together. See figure 1.10b. We can connect these boxes together to calculate p * (m + n). See Figure 1.10c. Assume we have an unlimited number of these boxes. Show how to connect them together to calculate:
   a. ax+b



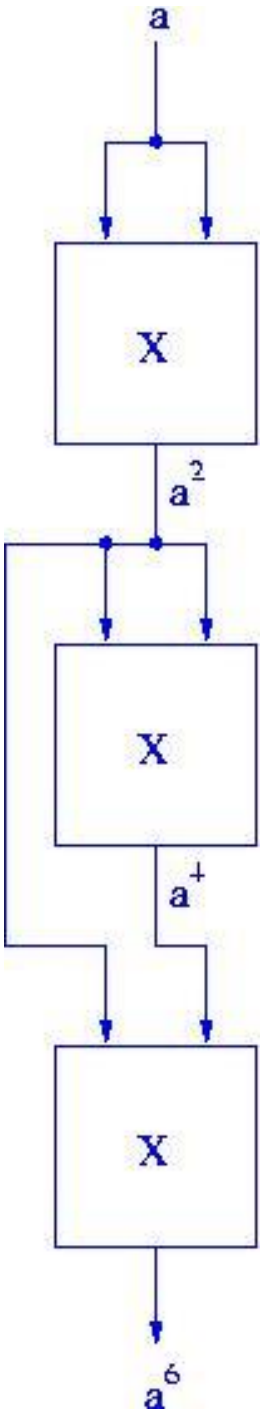
   b. The average of the four input numbers w, x, y, and z



c. $a^2 + 2ab + b^2$ (can you do it with one add box and one multiply box?)



d. a^6 (can you do it using only 3 multiply boxes?)



2. (2.3)
    a. Assume that there are about 400 students in your class. If every student is to be assigned a unique bit pattern, what is the minimum number of bits required to do this?

b. How many more students can be admitted to the class without requiring additional bits for each student's unique bit pattern?

112

3. (Adapted from 2.13)
Without changing their values, convert the following 2's complement binary numbers into 8-bit 2's complement numbers.
   a. 010110

      0001 0110

   b. 1101

      1111 1101

   c. 1111111000

      11111000

   d. 01

      00000001

4. (Adapted from 2.17)
Compute the following. Assume each operand is a 2's complement binary number.
   a. 01 + 1011

      1100

   b. 11 + 01010101

      01010100

   c. 0101 + 110

      0011

   d. 01 + 10

      11

5. Without changing their values, convert the following 8-bit 2's complement binary numbers into decimal numbers.
   a. 01010101

85

b. 10001101

-115

c. 10000000

-128

d. 11111111

-1

6. Express the value 0.3 in the 32-bit floating point format that we discussed in class today. Feel free to only show fraction bits [22:15], rather than all the fraction bits, [22:0]. Notation: The symbol [22:15] signifies all 8 bits from bit 22 to bit 15.

0 01111101 00110011

7. Convert the following floating point representation to its decimal equivalent:

1 10000010 10101001100000000000000

-13 19/64

8. Add the two hexadecimal 2's complement integers below:

$$\begin{array}{r} x90A \\ +\ x4123 \\ \hline \end{array}$$

F90A + 4123 = 3A2D

9. (Adapted from 2.50)

Perform the following logical operations. Express your answers in hexadecimal notation.

a. xABCD OR x9876

xBBFF

b. x1234 XOR x1234

x0000

c. xFEED AND (NOT(xBEEF))

x4000

10. (2.54)

Fill in the truth table for the equations given. The first line is done as an example.

Q1 = NOT (NOT(X) OR (X AND Y AND Z))
Q2 = NOT ((Y OR Z) AND (X AND Y AND Z))

X	Y	Z	Q1	Q2
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

11. (2.51)

What is the hexadecimal representation of the following numbers?

a. 25,675

x644B

b. 675.625 (i.e. $675\frac{5}{8}$), in the IEEE 754 floating point standard

x4428E800

c. The ASCII string: Hello

x48656C6C6F

1. What is the smallest positive normalized number that can be represented using the IEEE Floating Point standard?

2^(-126)

0 00000001 000000…000

What about the smallest positive integer that can NOT be represented using IEEE Floating Point standard?

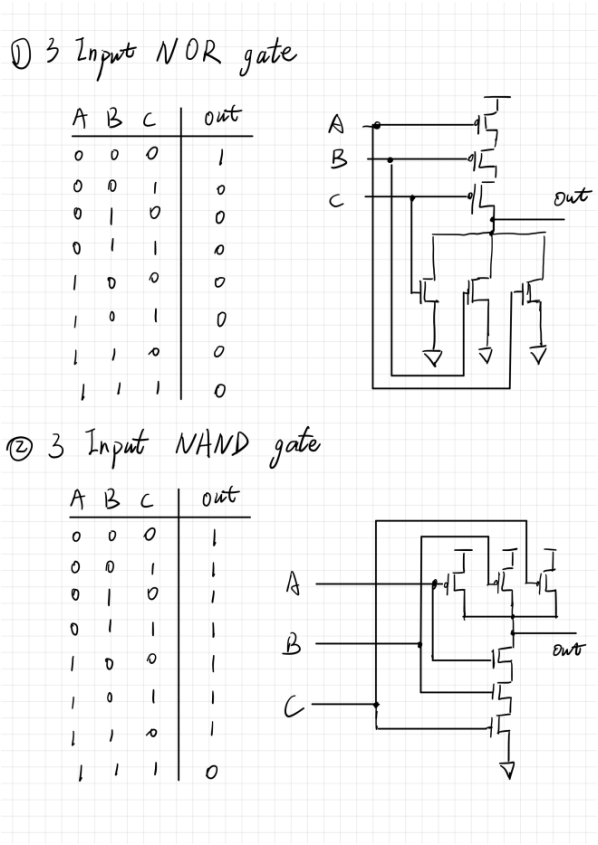
2. What is the largest positive number that can be represented in a 32 bit 2's complement scheme?

2,147,483,647

01111…111

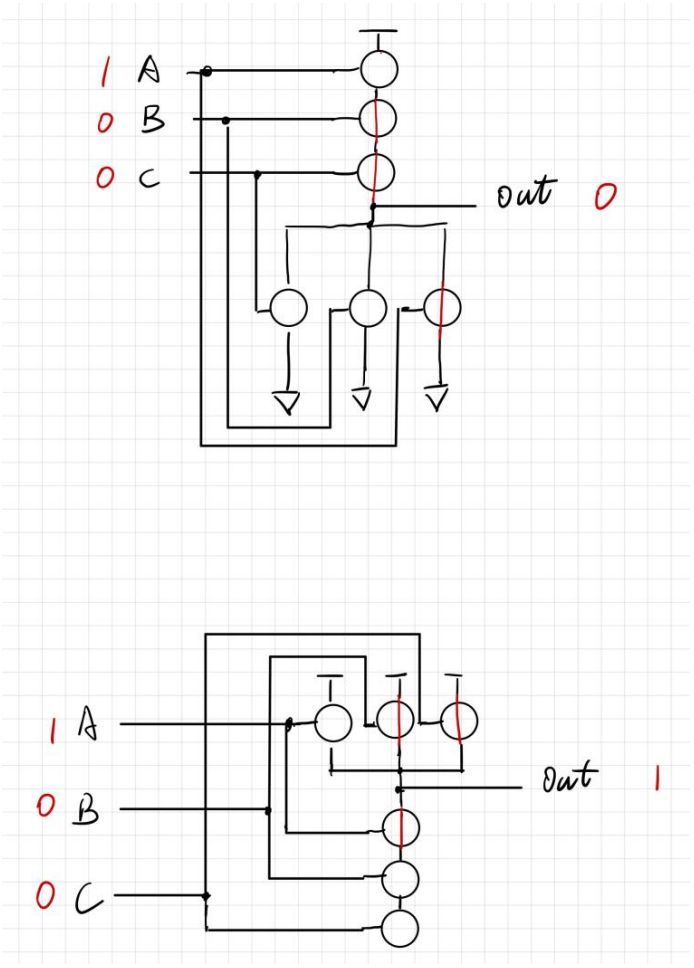
3.

a. (Adapted from 3.17) Draw a transistor-level diagram for a three-input NAND gate and a three-input NOR gate. Do this by extending the designs from following Figures 3.5a and 3.8a(NAND). (Figures can also be found in the book on pages 63 & 65 respectively).

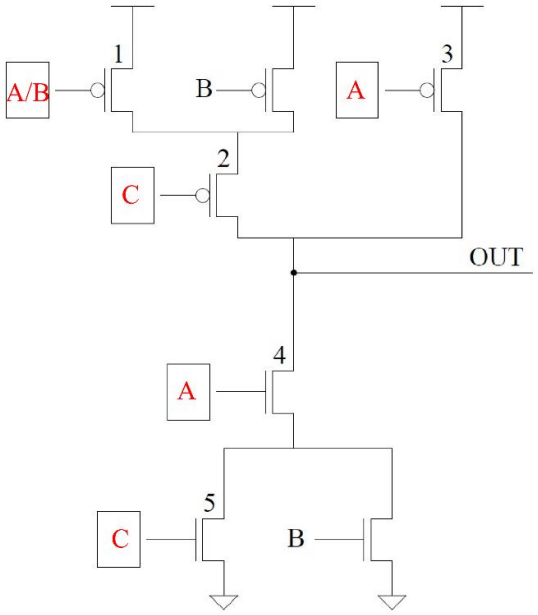


b. Replace the transistors in your diagrams from part (*a*) with either a wire or no wire to reflect the circuit's operation when the following inputs are applied:

A = 1, B = 0, C = 0



c. The transistor circuit shown below produces the accompanying truth table. The inputs to some of the gates of the transistors are not specified. Also, the outputs for some of the input combinations of the truth table are not specified. Complete both specifications. i.e., all transistors will have their gates properly labeled with either A, B, or C, and all rows of the truth table will have a 0 or 1 specified as the output.



A	B	C	OUT
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Figure 1

4. Shown below are several logical identities with one item missing in each. X represents the case where it can be replaced by either a 0 or a 1 and the identity will still hold. Your job: Fill in the blanks with either a 0, 1, or X.

   For example, in part a, the missing item is X. That is 0 OR 0 = 0 and 0 OR 1 = 1.

   a. 0 OR X = ___
      X
   b. 1 OR X = ___
      1
   c. 0 AND X = ___
      0
   d. 1 AND X = ___
      X
   e. __ XOR X = X
      0
   f. X XOR X = ___
      0

5. (3.25)
   Logic circuit 1 in Figure 3.36 (page 87 of the book) has inputs A, B, C. Logic circuit 2 in Figure 3.37 (page 87 of the book) has inputs A and B. Both logic circuits have an output D. There is a fundamental difference between the behavioral characteristics of these two circuits. What is it? *Hint*: What happens when the voltage at input A goes from 0 to 1 in both circuits?

   Figure 3.36 is a 2-input mux, which combinational logic i.e., D is the output of the circuit.
   Figure 3.37 is a storage element, which stores the data value previously stored in the latch.

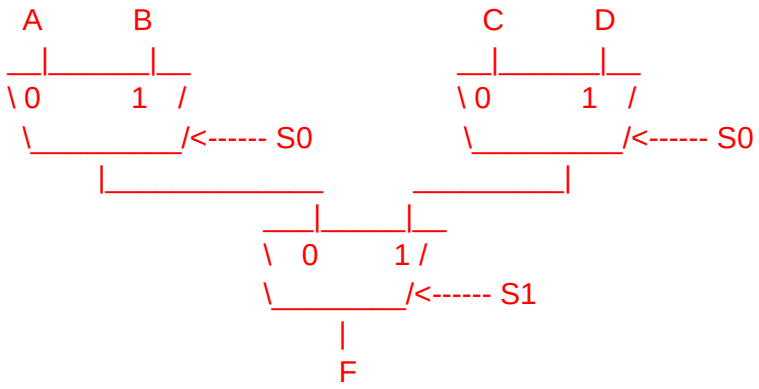
6. (Adapted from 3.28)

(1) Fill in the truth table of 4-to-1 mux:

S1	S0	A	B	C	D	OUT
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	0	1	1	0
0	0	0	1	0	0	0
0	0	0	1	0	1	0
0	0	0	1	1	0	0
0	0	0	1	1	1	0
0	0	1	0	0	0	1
0	0	1	0	0	1	1
0	0	1	0	1	0	1
0	0	1	0	1	1	1
0	0	1	1	0	0	1
0	0	1	1	0	1	1
0	0	1	1	1	0	1
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	0	0	0	1	0
0	1	0	0	1	0	0
0	1	0	0	1	1	0
0	1	0	1	0	0	1
0	1	0	1	0	1	1
0	1	0	1	1	0	1
0	1	0	1	1	1	1
0	1	1	0	0	0	0
0	1	1	0	0	1	0
0	1	1	0	1	0	0
0	1	1	0	1	1	0
0	1	1	1	0	0	1
0	1	1	1	0	1	1
0	1	1	1	1	0	1
0	1	1	1	1	1	1

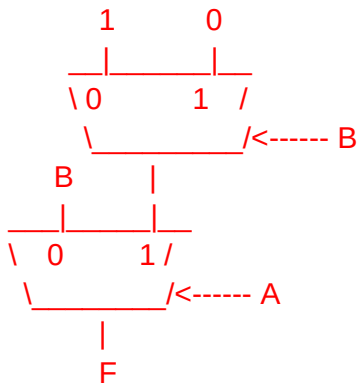
S1	S0	A	B	C	D	OUT
1	0	0	0	0	0	0
1	0	0	0	0	1	0
1	0	0	0	1	0	1
1	0	0	0	1	1	1
1	0	0	1	0	0	0
1	0	0	1	0	1	0
1	0	0	1	1	0	1
1	0	0	1	1	1	1
1	0	1	0	0	0	0
1	0	1	0	0	1	0
1	0	1	0	1	0	1
1	0	1	0	1	1	1
1	0	1	1	0	0	0
1	0	1	1	0	1	0
1	0	1	1	1	0	1
1	0	1	1	1	1	1
1	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	0
1	1	0	0	1	1	1
1	1	0	1	0	0	0
1	1	0	1	0	1	1
1	1	0	1	1	0	0
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	0	0	1	1
1	1	1	0	1	0	0
1	1	1	0	1	1	1
1	1	1	1	0	0	0
1	1	1	1	0	1	1
1	1	1	1	1	0	0
1	1	1	1	1	1	1

(2) Implement the 4-to-1 mux using only 2-to-1 muxes making sure to properly connect all of the terminals. Remember that you will have 4 inputs (A, B, C, and D), 2 control signals (S1 and S0), and 1 output (OUT).



You require 3 muxes. First, the input are A and B and the select line is S0. Second, inputs are C and D and the select line is also S0. Third, is a mux where both its inputs are the outputs of the first two muxes and select line is S1.

(3) Implement F = A XOR B using ONLY two 2-to-1 muxes. You are not allowed to use a NOT gate (A' and B' are not available).



7. (Adapted from 3.31)
Say the speed of a logic structure depends on the largest number of logic gates through which any of the inputs must propagate to reach an

output. Assume that a NOT, an AND, and an OR gate all count as one gate delay. For example, the propagation delay for a two-input decoder shown in Figure 3.11 is 2 because some inputs propagate through two gates.

a. What is the propagation delay for the two-input mux shown in Figure 3.12 (page 68)?

3

b. What is the propagation delay for the 4-bit adder shown in Figure 3.16 (page 71)?

12

c. Can you reduce the propagation delay for the circuit shown in Figure 3 by implementing the equation in a different way? If so, how?

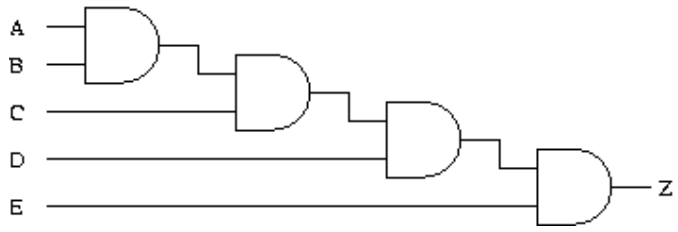


Figure 3

You can construct a tree-like structure.

E = ((A AND B) AND (C AND D)) AND E)

8. (3.32)
Recall that the adder was built with individual "slices" that produced a sum bit and carryout bit based on the two operand bits A and B and the carryin bit. We called such an element a full-adder. Suppose we have a 3-to-8 decoder and two six-input OR gates, as shown in Figure 3 below. Can we connect them so that we have a full-adder? If so, please do. (*Hint*: If an input to an OR gate is not needed, we can simply put an input 0 on it and it will have no effect on anything. For example, see the figure below.)

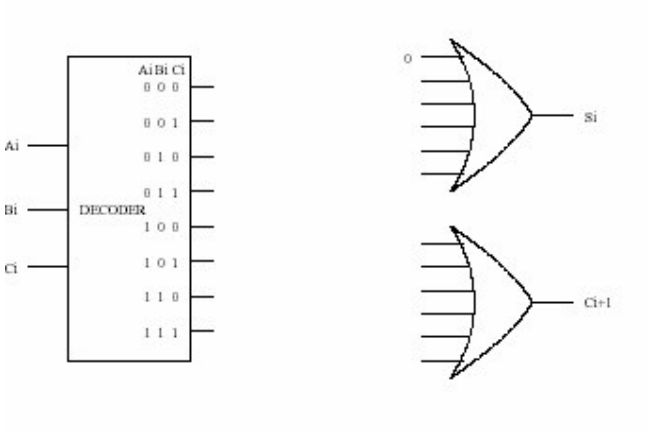
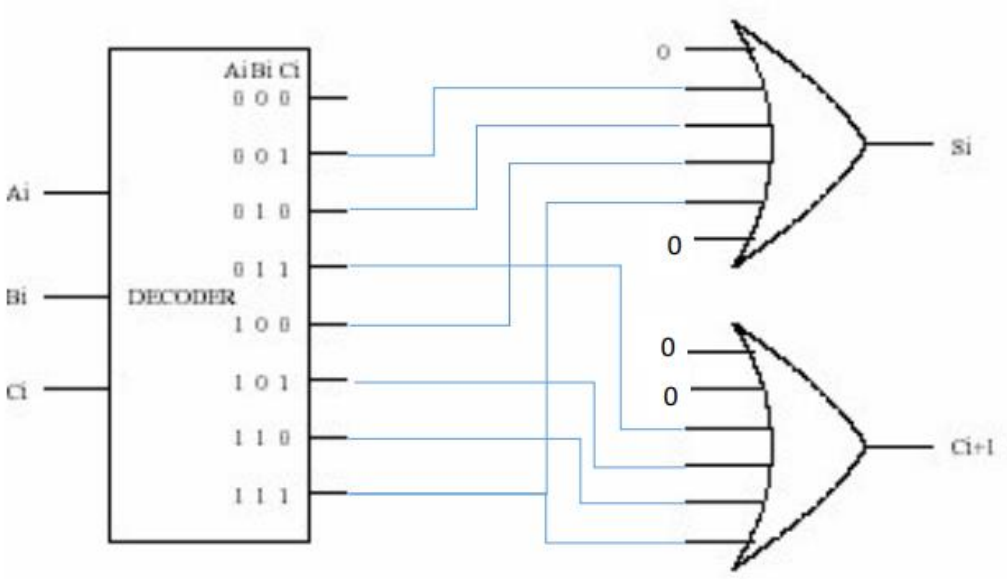


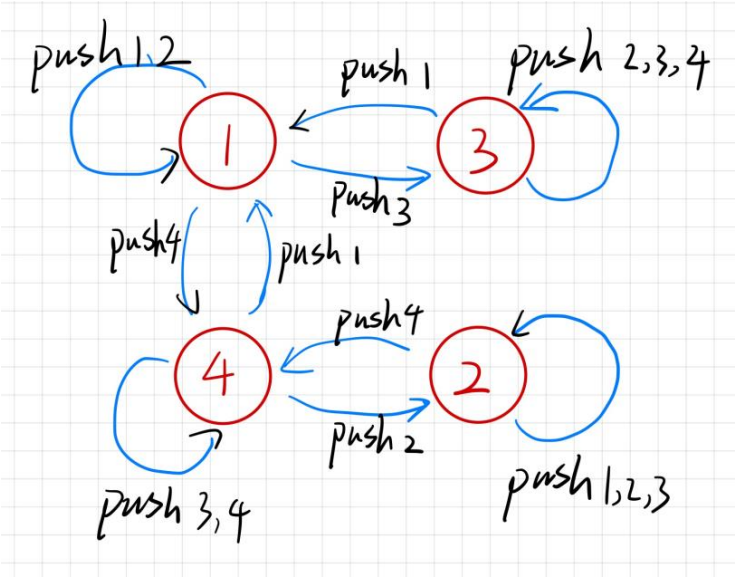
Figure 3



9. We wish to design a controller for an elevator such that if you push a button for a desired floor, the controller will output the floor number that the elevator should go to. However, to deter lazy people from going up or down one floor, if you push the button for the next floor (up or down), the elevator will stay on its current floor. If you push the button for the same floor that you're currently on, the controller will output the current floor number. There are four floors in the building.

   Your job:

   a. Draw the state diagram of the elevator scheduling.



b. Construct a complete truth table for the elevator controller. It is not necessary to draw the logic here; the truth table is sufficient.

Since there are four floors, you will need 2 bits to represent a floor. Let the logic variable C[1:0] represent the current floor, R[1:0] represent the requested floor, and D[1:0] represent the floor the elevator should go to given a current floor and a requested floor. Shown below is the truth table for this combinational logic circuit.

C1	C0	R1	R0	D1	D0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	1
1	1	1	1	1	1

10.

A logic circuit consisting of 6 gated D latches and 1 inverter is shown below:

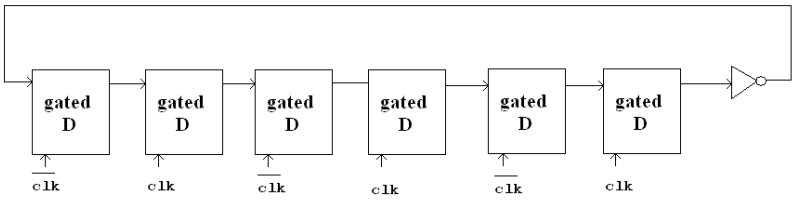


Figure 5

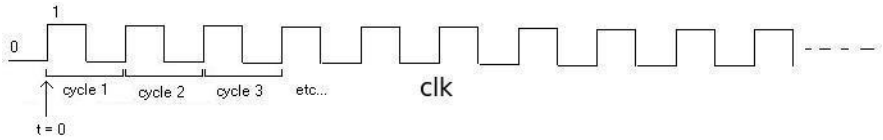


Figure 6

Let the state of the circuit be defined by the state of the 6 D latches. Assume initially the state is 000000 and clk starts at the point labeled t0.

Question: What is the state after 50 cyles. How many cycles does it take for a specific state to show up again?

Every 6 clock cycles a pattern repeats. A and B represent the first half and the second half of each clock cycle respectively.

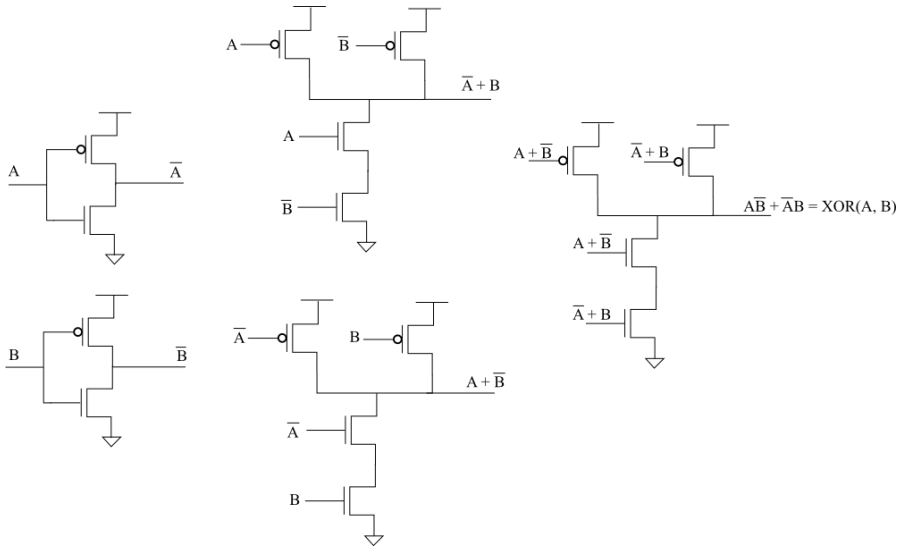
Cycle1 A: 000000
Cycle1 B: 100000
Cycle2 A: 110000
Cycle2 B: 111000
Cycle3 A: 111100
Cycle3 B: 111110
Cycle4 A: 111111
Cycle4 B: 011111
Cycle5 A: 001111
Cycle5 B: 000111
Cycle6 A: 000011
Cycle6 B: 000001

Cycle7 A: 000000

Because 50 = 6*8+2 after 50 cycles the state will be the same as after 2 cycles. It will be in state 111000 after 50 cycles

11.

Draw the transistor level circuit of a 2 input XOR gate



12. (Adapted from 3.36)

A comparator circuit has two 1-bit inputs, A and B, and three 1-bit outputs, G (greater), E (equal), and L (less than). Refer to figures 3.43 and 3.44 on page 106 in the book for this problem..

a. Draw the truth table for a 1-bit comparator.

A	B	G	E	L
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

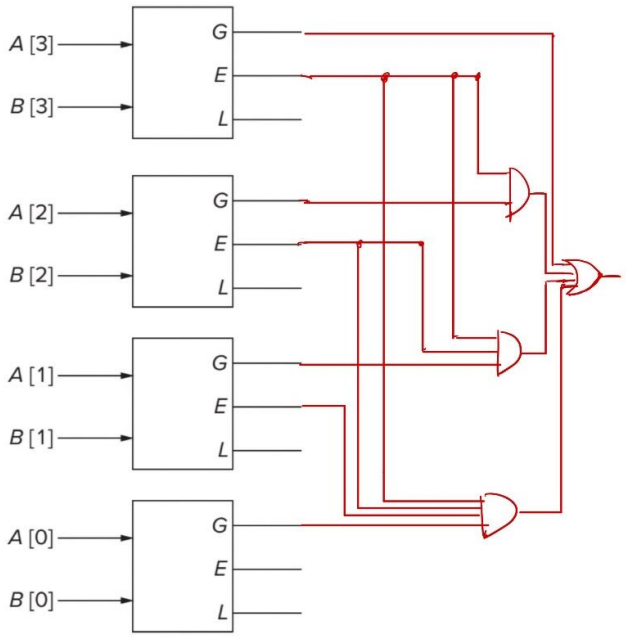
b. Implement G, E and L for a 1-bit comparator using AND, OR, and NOT gates.

G = AB' , L = A'B , E = A'B' + AB

c. Figure 3.44 performs one-bit comparisons of the corresponding bits of two unsigned integer A[3:0] and B[3:0]. Using the 12 one-bit results of these 4 one-bit comparators, construct a logic

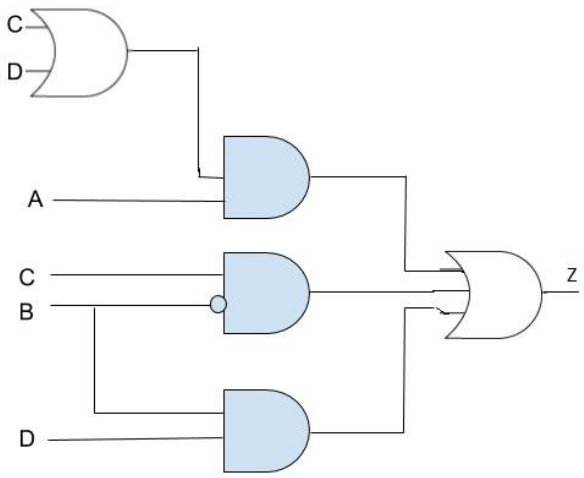
circuit to output a 1 if unsigned integer A is larger than unsigned integer B (the logic circuit should output 0 otherwise). The inputs to your logic circuit are the outputs of the 4 one-bit comparators and should be labeled G[3], E[3], L[3], G[2], E[2], L[2], ... L[0]. (Hint: You may not need to use all 12 inputs.)

Y = G[3] + E[3]G[2] + E[3]E[2]G[1] + E[3]E[2]E[1]G[0]



13. One of Zhang San's students is always late to meetings, so Professor Zhang San wants you to design an alarm clock to help his student be on time. Your job is to design a logic circuit whose output Z is equal to 1 when the alarm clock should go off. The circuit will receive four input variables (A, B, C, D) that answer four different yes/no question (1=yes, 0=no):
    A <= Is it going to be sunny today?
    B <= Is it the weekend?
    C <= Is it 7:00am?
    D <= Is it 9:00am?

    Zhang San wants the alarm clock to go off if it's sunny and it's either 7:00am or 9:00am. The alarm clock should go off if it's the weekend and it's 9:00am. The alarm clock should also go off if it's not the weekend and it's 7:00am. Write the truth table and draw a gate-level diagram that performs this logic.



A	B	C	D	ALARM
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	x
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	x
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	x
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	x

14. Prove that NAND is logically complete.

NOT: A NAND A => NOT A

AND: (A NAND B) NAND (A NAND B) = NOT (A NAND B) => A AND B

OR: NOT(NOT(A) AND NOT(B)) => A OR B

1. 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, 50
   6) Possesion: 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

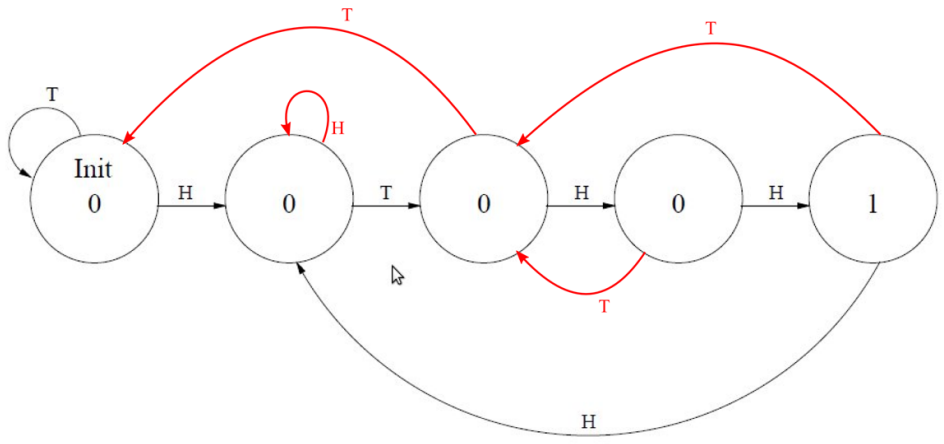
(c) Why might the method of part b be a better way to specify the state than the method of part a?

The assignments in (b) are easier to decode

2. Shown below is a partially completed state diagram of a finite state machine that takes an input string of H (heads) ant 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

Figure 4



For example,

if the input string is: H H H H H T H H T H H H H H T H H T,
the output would be: 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0.

Note that the 8^{th} coin toss (H) is part of two HTHH sequences.

b) If this state machine is implemented with a sequential logic circuit how many state variables will be needed?

3 bits

3. (3.37)

If a particular computer has 8 byte addressability and a 8 bit address space, how many bytes of memory does that computer have?

Number of byes = address space x adressability. 2^8 x

2^3 = 2^11 = 2048 bytes

4. (3.33)

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 third memory location, what must the values of *A[1:0]* and *WE* be?

To read from the third location `A[1:0]` should be 10, 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.

5. 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.

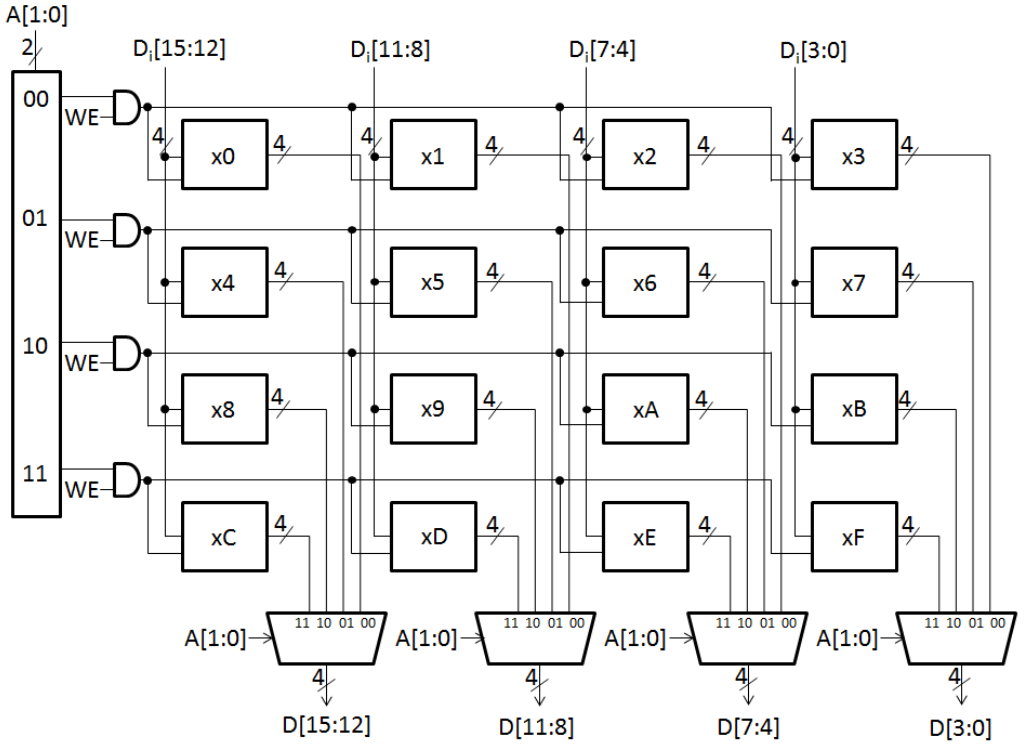


Figure 3: 2^2-by-16 bit memory

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]	Di[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

6. (4.8)
Suppose a 32-bit instruction has the following format:

OPCODE	DR	SR1	SR2	UNUSED

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*?

   255 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 ununsed bits

7. *A State Diagam*

We wish to invent a two-person game, which we will call XandY that can be played on the computer. Your job in this problem is 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 intial 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 transistions 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.

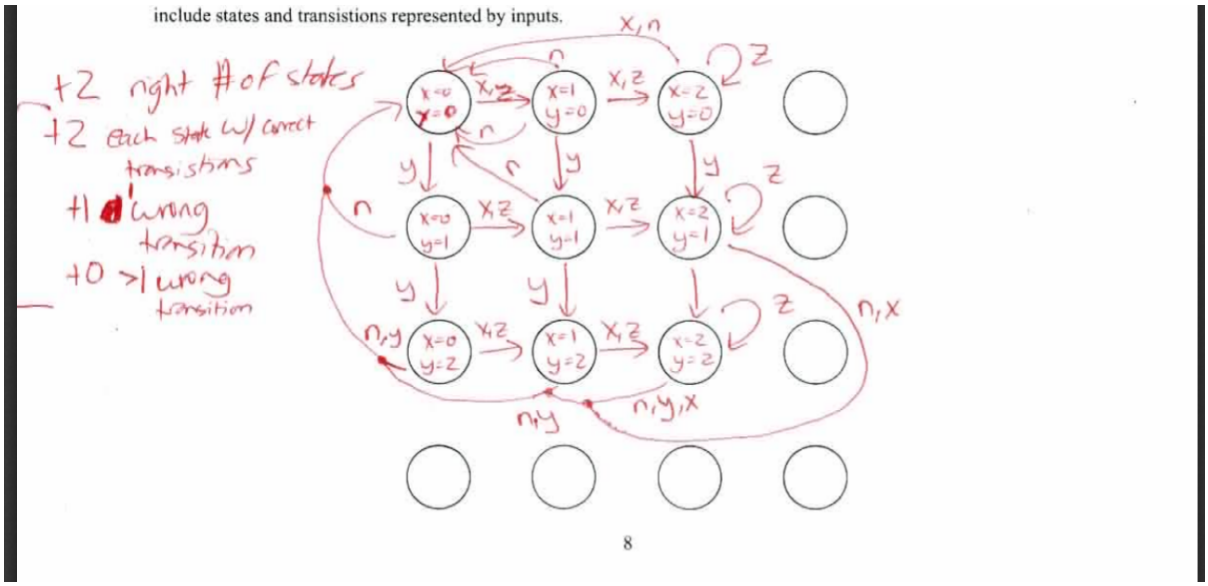
Important rule: If the number of X's or Y's reaches a count of 3, the current round is terminated and another round is started. When a round starts, its state is X=0, Y=0.

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 cannot have X=3, because then the round would be finished, and we would have started a *new* current round.

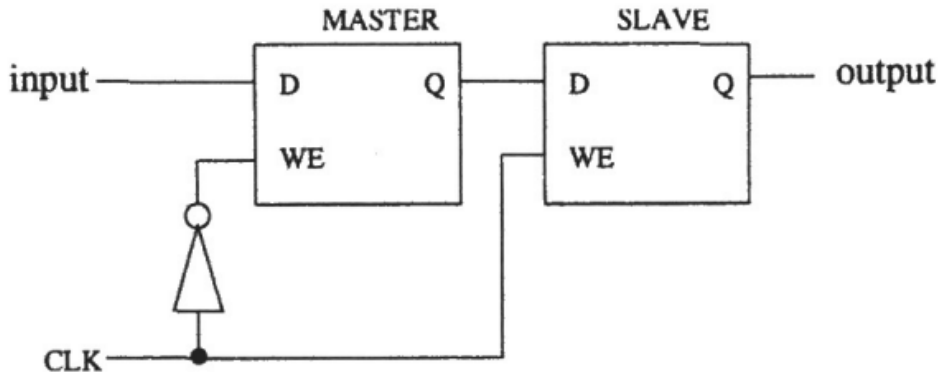
On the diagram below, label each state. For each state draw an arrow showing the transition to the next state that would occur for each of the four inputs. (We have provided sixteen states. You will not need all of them. Use only as many as you need).

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

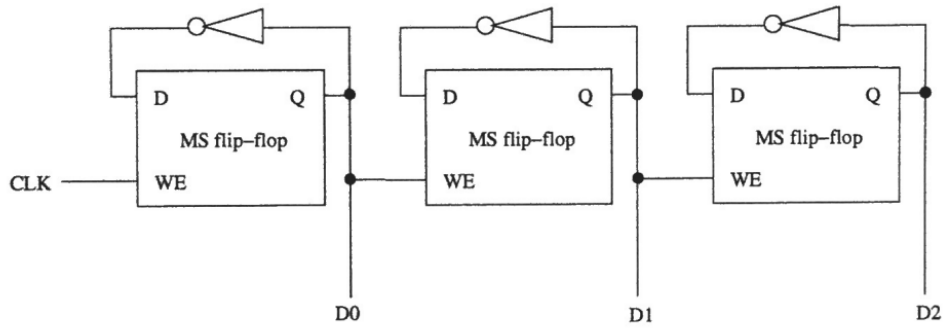


8. *Trying Out Flip-Flops*

The we introduced in class is shown below.



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 flipflops.



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)

	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 positiv "edge" of D_1

inverts on positive "edge" of D_0

inverts on positive "edge" of clock

"edges" in bold

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

D_2, D_1, D_0 act as a decrementing counter

Homework04

1. 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 011 000000001
   0001 100 100 1 00001
   1111 0000 0010 0101

   R2<R1 R4=1

   R2>=R1 R4=0

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

   0101 000 000 1 00000
   0101 101 001 1 00001
   0000 010 000000001
   0001 000 000 1 00001
   1111 0000 0010 0101
   R5 is even R0=0
   R5 is odd R0=1

3. (Adapted from 5.31) The following diagram shows a snapshot of the 8 registers of the LC-3 before and after the instruction at location x1000 is executed. Fill in the bits of the instruction at location x1000.

Register	Before	After
R0	x0000	x0000
R1	x1111	x1111
R2	x2222	x2222
R3	x3333	x3333
R4	x4444	x4444

R5	x5555	xFFF8
R6	x6666	x6666
R7	x7777	x7777

Memory Location	Value
x1000	0001 _101 000 1 11000__

4. The memory locations x3000 to x3007 contain the values as shown in the table below. Assume the memory contents below are loaded into the simulator and the PC has been set to point to location x3000. Assume that a breakpoint has been placed to the left of the HALT instruction (i.e. at location x3006 which contains 1111 0000 0010 0101). Assume that before the program is run, each of the 8 registers has the value x0000 and the NZP bits are 010.

Memory Location	Value
x3000	0101000000100000
x3001	0001000000100101
x3002	0010001000000100
x3003	0001000000000000
x3004	0001001001111111
x3005	0000001111111101
x3006	1111000000100101
x3007	0000000000000100

a. In no more than 15 words, summarize what this program will do when the Run button is pushed in the simulator.

   Hint: What relationship is there between the value loaded from memory and the final value in R0 after the program has completed?

   5 is put in R0 and shifted left the value at location x3007 times

b. What are the contents of the PC, the 8 general purpose registers (R0-R7), and the N, Z, and P condition code registers after the program completes?

   PC x3006

   R0 x0050

   R1 x0000

   R2 x0000

   R3 x0000

   R4 x0000

   R5 x0000

   R6 x0000

   R7 x0000

   N 0

   Z 1

   P 0

c. What is the total number of CPU clock cycles that this program will take to execute until it reaches the breakpoint?

   Note: You should refer to the state machine (pg 702) to determine how many cycles an instruction takes. Assume each state that access memory takes 5 cycles to complete and every other state takes 1 cycle to execute. States that check for ACV also take 1 cycle to execute

Memory Location	Value	Instruction	Cycles takes to exectue once	number of times executed	Total Cycles for instruction
X3000	0101000000100000	AND	10	1	10
X3001	0001000000100101	ADD	10	1	10
X3002	0010001000000100	LD	17	1	17
X3003	0001000000000000	ADD	10	4	40
X3004	0001001001111111	ADD	10	4	40
X3005	0000001111111101	Branch	10 if not taken 11 if taken	3 times taken 1 time not taken	43

Total Cycles 10+10+17+40+40+43 = 160

5. What does the following program do (in 15 words or fewer)? The PC is initially at x3000. (Assume that before the program is run,R0 has the value x0000.)

Memory Location	Value
x3000	0001 000 000 1 10000
x3001	0010 001 011111110
x3002	0000 010 000000100
x3003	0000 011 000000001

x3004	0001 000 000 1 00001
x3005	0001 001 001 000 001
x3006	0000 111 111111011
X3007	1001 000 000 111111
X3008	0001 000 000 1 00001
x3009	1111 0000 0010 0101

Counts the number of bits that are set to 0 in the word at x3100

6. Prior to executing the following program, memory locations x3100 through x4000 are initialized to random values, exactly one of which is negative. The following program finds the address of the negative value, and stores that address into memory location x3050. Two instructions are missing. Fill in the missing instructions to complete the program. The PC is initially at x3000.

Memory Location	**Value**
x3000	1110 000 011111111
x3001	**0110 001 000 000000**
x3002	**0000 100 000000010**
x3003	0001 000 000 1 00001
x3004	0000 111 111111100
x3005	0011 000 001001010

x3006	1111 0000 0010 0101

7. The LC-3 has just finished executing a large program. A careful examination of each clock cycle reveals that the number of executed store instructions (ST, STR, and STI) is greater than the number of executed load instructions (LD, LDR, and LDI). However, the number of memory write accesses is less than the number of memory read accesses, *excluding instruction fetches*. How can that be? Be sure to specify which instructions may account for the discrepancy

   A large number of LDI instructions (two read accesses) and STI instructions (one read access and one write access) could account for this discrepancy.

8. We would like to have an instruction that does nothing. Many ISAs actually have an opcode devote to doing nothing. It's usually called NOP, for NO OPERATION. The instruction is fetched, decoded, and executed. The execution phase is to do nothing! Which of the following three instructions could be used for NOP and have the program still work correctly?
   a) 0001 001 001 1 00000
   b) 0000 111 000000001
   c) 0000 000 000000000

   What does the instruction(s) couldn't be used for NOP do that other do not do?

   a) Add R1, R1, #0 => differs from a NOP in that it sets the CC's.
   b) BRnzp #1 => Unconditionally branches to one after the next address in the PC. Therefore no, this instruction is not the same as NOP.
   c) Branch that is never taken. Yes same as NOP.

9. The LC-3 does not have an opcode for the logical function OR. The four instruction sequence below performs the OR of the contents of register 1 and register 2 and puts the result in register 3. Fill in the two missing instructions so that the four instruction sequence will do the job.
   1) 1001 100 001 111111
   2) 1001 101 010 111111
   3) 0101 110 100 000 101

4) 1001 011 110 111111

Chapter 7&8

1. consider the following program written in LC-3 assembly language:

```
                .ORIG x3000
                AND R5, R5, #0
                LEA R0, ARRAY
                LD  R1,  N
                LDR R2,  R0,   #0
                NOT R2, R2
                ADD R2, R2,  #1
LOOP        LDR  R3,  R0,  #0
                ADD  R3,  R3,  R2
                BRnp   DONE
                ADD  R0,  R0, #1
                ADD  R1,  R1,  #-1
                BRp LOOP
                ADD    R5,    R5,  #1
DONE     ST   R5,  OUTPUT
                HALT
ARRAY       .BLKW  #20
N             .FILL   #20
OUTPUT  .BLKW  #1
                .END
```

What must be the case for 1 to be stored in OUTPUT? Answer in 15 words or fewer.

When all elements in array are same.

2.An Aggie tried to write a recursive subroutine which, when given an integer n, return the sum of the first n positive integers. For example, for n = 4, the subroutine returns 10 (i.e., 1 + 2 + 3 + 4). The subroutine takes the argument n in R0 and returns the sum in R0.

```
1                               SUM     ADD R6, R6, #-1
2                                       STR R7, R6, #0
3                                       ADD R6, R6, #-1
4                                       STR R1, R6, #0
5                                       ADD R1, R0, #0
6                                       ADD R0, R0, #-1
7                                       JSR SUM
8                                       ADD R0, R0, R1
9                                       LDR R1, R6, #0
10                                      ADD R6, R6, #1
11                                      LDR R7, R6, #0
12                                      ADD R6, R6, #1
13                                      RET
```

Unfortunately, the recursive subroutine does not work. What is the problem? Explain in 15 words or fewer.And modify the program to make it work.

There is no base case.

```
1                               SUM     ADD R6, R6, #-1
2                                       STR R7, R6, #0
3                                       ADD R6, R6, #-1
4                                       STR R1, R6, #0
5                                       ADD R1, R0, #0
6                                       ADD R0, R0, #-1
7                                       BRnz NEXT
8                                       JSR SUM
9                               NEXT    ADD R0, R0, R1
10                                      LDR R1, R6, #0
11                                      ADD R6, R6, #1
12                                      LDR R7, R6, #0
13                                      ADD R6, R6, #1
14                                      RET
```

3.Memory locations x5000 to x5FFF contain 2's complement integers. What does the following program do?

```
1           .ORIG x3000
2            LD  R1, ARRAY
3            LD  R2, LENGTH
4            AND R3, R3, #0
```

```
5    AGAIN   LDR R0, R1, #0
6            AND R0, R0, #1
7            BRz SKIP
8            ADD R3, R3, #1
9    SKIP    ADD R1, R1, #1
10           ADD R2, R2, #-1
11           BRp AGAIN
12           HALT
13   ARRAY   .FILL x5000
14   LENGTH .FILL x1000
15           .END
```

Please write your answer in the box below. Your answer must contain at most 15 words. Any words after the first 15 will NOT be considered in grading this problem.

count the number of odd numbers in the array.

4.It is easier to identify borders between cities on a map if a adjacent cities are colored with the different colors. For example, in a map of Texas, one would not color Austin and Pflugerville with the same color, since dong so would obscure the border bewteen the two cities.

Shown below is the recursive subroutine EXAMINE. EXAMINE examines the data structure representing a map to see if any pair of adjacent cities have the same color. Each node in the data structure contains the city's color and the addresses of the cities it borders. If no pair of adjacent cities have the same color, EXAMINE returns the value 0 in R1. If at least one pair of adjacent cities have the same color, EXAMINE returns the value 1 in R1. The main program supplies the address of a node representing one of the cities in R0 before executing JSR EXAMINE.

```
1           .ORIG x4000
2    EXAMINE ADD R6, R6, #-1
3            STR R0, R6, #0
4            ADD R6, R6, #-1
5            STR R2, R6, #0
6            ADD R6, R6, #-1
7            STR R3, R6, #0
8            ADD R6, R6, #-1
9            STR R7, R6, #0
10
11           AND R1, R1, #0     ; Initialize output R1 to 0
12           LDR R7, R0, #0
13           BRn RESTORE        ; Skip this node if it has already been visited
```

```
14
15          LD  R7, BREADCRUMB
16          STR R7, R0, #0    ; Mark this node as visited
17          LDR R2, R0, #1    ; R2 = color of current node
18          ADD R3, R0, #2
19
20  AGAIN   LDR R0, R3, #0    ; R0 = neighbor node address
21          BRz RESTORE
22          LDR R7, R0, #1
23          NOT R7, R7        ; <-- Breakpoint here
24          ADD R7, R7, #1
25          ADD R7, R2, R7    ; Compare current color to neighbor's color
26          BRz BAD
27          JSR EXAMINE       ; Recursively examine the coloring of next neighbor
28          ADD R1, R1, #0
29          BRp RESTORE       ; If neighbor returns R1=1, this node should return R1=1
30          ADD R3, R3, #1
31          BR  AGAIN         ; Try next neighbor
32
33  BAD     ADD R1, R1, #1
34  RESTORE LDR R7, R6, #0
35          ADD R6, R6, #1
36          LDR R3, R6, #0
37          ADD R6, R6, #1
38          LDR R2, R6, #0
39          ADD R6, R6, #1
40          LDR R0, R6, #0
41          ADD R6, R6, #1
42          RET
43
44  BREADCRUMB .FILL x8000
45          .END
```

Your job is to construct the data structure representing a particular map. Before executing JSR EXAMINE, R0 is set to x6100 (the address of one of the nodes), and a breakpoint is set at x4012. The table below shows relevant information collected each time the breakpoint was encountered during the running of EXAMINE.

PC	R0	R2	R7
x4012	x6200	x0042	x0052
x4012	x6100	x0052	x0042
x4012	x6300	x0052	x0047
x4012	x6200	x0047	x0052
x4012	x6400	x0047	x0052
x4012	x6100	x0052	x0042
x4012	x6300	x0052	x0047
x4012	x6500	x0052	x0047
x4012	x6100	x0047	x0042
x4012	x6200	x0047	x0052
x4012	x6400	x0047	x0052
x4012	x6500	x0052	x0047
x4012	x6400	x0042	x0052
x4012	x6500	x0042	x0047

Construct the data structure for the particular map that corresponds to the relevant information obtained from the break- points. Note: We are asking you to construct the data structure as it exists AFTER the recursive subroutine has executed.

x6100	x8000	x6300	X8000	x6500	X8000
x6101	X0042	x6301	X0047	x6501	X0047
x6102	X6200	x6302	x6200	x6502	X6100
x6103	X6400	x6303	x6400	x6503	X6200
x6104	x6500	x6304	X0000	x6504	X6400
x6105	X0000	X6305		x6505	X0000
x6106		x6306		x6506	
x6200	X8000	x6400	x8000		
x6201	X0052	x6401	x0052		
x6202	X6100	x6402	x6100		
x6203	X6300	x6403	x6300		
x6204	X6500	x6404	x6500		
x6205	X0000	x6405	X0000		
x6206		x6406			

5. The following program, after you insert the two missing instructions, will examine a list of positive integers stored in consecutive sequential memory locations and store the smallest one in location x4000. The number of integers in the list is contained in memory location x4001. The list itself starts at memory location x4002. Assume the list is not empty (i.e., the contents of x4001 is not zero.)

```
1                .ORIG x3000
2                LDI R1, SIZE
3                LD  R2, LISTPOINTER
4                LDR R0, R2, #0
5                ADD R1, R1, #-1
6                BRz ALMOSTDONE        ;Only one element in the list
7   AGAIN        ADD R2,R2,#1
8
9                LDR R3,R2,#0
10               NOT R4,R3
11               ADD R4,R4,#1
12               ADD R4,R0,R4
13               BRnz  SKIP
14               ADD R0,R3,#0
```

```
SKIP            ADD R1,R1,#-1
                BRp AGAIN

ALMOSTDONE      LD R5,MIN
                STR R0,R5,#0
                HALT

MIN             .FILL x4000
SIZE            .FILL x4001
LISTPOINTER     .FILL x4002
                .END
```

Your job: Insert the two the missing instructions.

6.Your job in this problem will be to add the missing instructions to a program that detects palindromes. Recall a palin- drome is a string of characters that are identical when read from left to right or from right to left. For example, racecar and 112282211. In this program, we will have no spaces and no capital letters in our input string – just a string of lower case letters.

The program will make use of both a stack and a queue. The subroutines for accessing the stack and queue are shown below. Recall that elements are PUSHed (added) and POPped (removed) from the stack. Elements are ENQUEUEd (added) to the back of a queue, and DEQUEUEd (removed) from the front of the queue.

```
            .ORIG x3050
PUSH        ADD R6, R6, #-1
            STR R0, R6, #0
            RET
POP         LDR R0, R6, #0
            ADD R6, R6, #1
            RET
STACK       .BLKW #20
            .END


            .ORIG x3080
ENQUEUE     ADD R5, R5, #1
            STR R0, R5, #0
            RET
DEQUEUE     LDR R0, R4, #0
            ADD R4, R4, #1
            RET
```

```
QUEUE      .BLKW #20
           .END
```

The program is carried out in two phases. Phase 1 enables a user to input a character string one keyboard character at a time. The character string is terminated when the user types the enter key (line feed). In Phase 1, the ASCII code of each character input is pushed on a stack, and its negative value is inserted at the back of a queue. Inserting an element at the back of a queue we call enqueuing.

In Phase 2, the characters on the stack and in the queue are examined by removing them, one by one from their re- spective data structures (i.e., stack and queue). If the string is a palindrome, the program stores a 1 in memory location RESULT. If not, the program stores a zero in memory location RESULT. The PUSH and POP routines for the stack as well as the ENQUEUE and DEQUEUE routines for the queue are shown below. You may assume the user never inputs more than 20 characters.

The program for detecting palindromes (with some instructions missing) .

Your job is to fill in the missing instructions.

```
           .ORIG X3000
           LEA   R4, QUEUE
           LEA   R5, QUEUE
           ADD   R5, R5, #-1
           LEA   R6, ENQUEUE       ;Initialize SP
           LD    R1, ENTER
           AND   R3, R3, #0
;
           LEA R0,PROMPT
           TRAP x22
PHASE1     TRAP x20
           ADD R2,R0,R1
           BRz PHASE2
           JSR PUSH
           NOT R0,R0
           ADD R0,R0,#1
           JSR ENQUEUE
           ADD R3, R3, #1
           BRnzp PHASE1
;
PHASE2     JSR POP
           ADD R1,R0,#0
           JSR DEQUEUE
           ADD R1, R0, R1
           BRnp  FALSE
           ADD R3,R3,#-1
           BRz TURE
           BRnzp PHASE2
;
TRUE       AND R0, R0, #0
```

```
31              ADD R0, R0, #1
32              ST  R0, RESULT
33              HALT
34  FALSE       AND R0, R0, #0
35              ST  R0, RESULT
36              HALT
37  RESULT      .BLKW #1
38  ENTER       .FILL x-0A
39  PROMPT      .STRING "Enter an input string"
40              .END
```

More problems approaching!


7850
75
75
75
125
25
25
0
25
150
50
Menu
More zombies approaching!

HW06_solution

1.The following program is supposed to print the number 5 on the screen. It does not work. Why? Answer in no more than ten words, please.

```
1        .ORIG  x3000
2        JSR    A
3        OUT             ;TRAP x21
4        BRnzp  DONE
5  A     AND    R0,R0,#0
6        ADD    R0,R0,#5
7        JSR    B
8        RET
9  DONE  HALT
10 ASCII .FILL  x0030
11 B     LD     R1,ASCII
12       ADD    R0,R0,R1
13       RET
14       .END
```

Need to save R7 so 1st service routine can return. Second RET overwrites the first RET value.

2.The following LC-3 program is assembled and then executed. There are no assemble time or run-time errors. What is the output of this program? Assume all registers are initialized to 0 before the program executes.

```
1         .ORIG x3000
2         ST R0, #6 ; x3007
3         LEA R0, LABEL
4         TRAP x22
5         TRAP x25
6  LABEL  .STRINGZ "FUNKY"
7  LABEL2 .STRINGZ "HELLO WORLD"
8         .END
```

FUN

3.The following nonsense program is assembled and executed.

```
        .ORIG x4000
        LD R2,BOBO
        LD R3,SAM
AGAIN   ADD R3,R3,R2
        ADD R2,R2,#-1
        BRnzp SAM
BOBO    .STRINGZ "Why are you asking me this?"
SAM     BRnp AGAIN
        TRAP x25
        .BLKW 5
JOE     .FILL x7777
        .END
```

How many times is the loop executed? When the program halts, what is the value in R3? (If you do not want to the arithmetic, it is okay to answer this with a mathematical expression.)

Work: BOBO is length `28 (27 + 1 for null)`. BRnp AGAIN in binary is `0000 101 #-32 = 0000 101 1 1110 0000 = x0BE0`. R3 holds `x0BE0`. R2 starts with the value of W which is `x57`. R. The loop executes `57` times. The final value of R3 is `x0BE0 + (x57 + x1) * x57 / x2 = x0BE0 + x0EF4 = x1AD4` or `#6868`. Note that `x0BE0` is `#3040`.

4.The program below, when complete, should print the following to the monitor:

ABCFGH

Insert instructions at (a)–(d) that will complete the program.

```
        .ORIG x3000
        LEA   R1, TESTOUT
BACK_1  LDR   R0, R1, #0
        BRz   NEXT_1
        TRAP  x21
        ADD   R1, R1, #1  ;(a)
        BRnzp BACK_1
        ;
NEXT_1  LEA   R1, TESTOUT
BACK_2  LDR   R0, R1, #0
        BRz   NEXT_2
        JSR   SUB_1
        ADD   R1, R1, #1
        BRnzp BACK_2
        ;
NEXT_2  HALT              ;(b)
        ;
SUB_1   ADD   R0, R0, #5  ;(c)
```

```
K       LDI   R2, DSR
        BRzp  K           ;(d)
        STI   R0, DDR
        RET
DSR     .FILL xFE04
DDR     .FILL xFE06
TESTOUT .STRINGZ "ABC"
        .END
```

5.Shown below is a partially constructed program. The program asks the user his/her name and stores the sentence "Hello, name" as a string starting from the memory location indicated by the symbol HELLO. The program then outputs that sentence to the screen. The program assumes that the user has finished entering his/her name when he/she presses the Enter key, whose ASCII code is x0A. The name is restricted to be not more than 25 characters.

Assuming that the user enters Onur followed by a carriage return when prompted to enter his/her name, the output of the program looks exactly like:

Please enter your name: Onur
Hello, Onur

Insert instructions at (a)–(d) that will complete the program.

```
          .ORIG x3000
          LEA    R1,HELLO
AGAIN     LDR    R2,R1,#0
          BRz    NEXT
          ADD    R1,R1,#1
          BR     AGAIN
NEXT      LEA    R0,PROMPT
          TRAP   x22           ; PUTS
          LD     R3 NEGENTER     ;a
AGAIN2    TRAP   x20           ; GETC
          TRAP   x21           ; OUT
          ADD    R2,R0,R3
          BRz    CONT
          STR    R0,R1,#0        ;b
          ADD    R1,R1,#1        ;c
          BR     AGAIN2
CONT      AND    R2,R2,#0
          STR    R2,R1,#1        ;d
          LEA    R0, HELLO
          TRAP   x22           ; PUTS
          TRAP   x25           ; HALT
NEGENTER  .FILL  xFFF6         ; -xoA
PROMPT    .STRINGZ "Please enter your name: "
HELLO     .STRINGZ "Hello, "
```

```
25          .BLKW #25
26          .END
```