Midterm 2

NAME:_____ SID_____

Instructions

Read all of the instructions and all of the questions before beginning the exam.

There are 5 problems in this exam. The total score is 100 points. Points are given next to each problem to help you allocate time. Do not spend all your time on one problem.

Unless otherwise noted on a particular problem, you must show your work in the space provided, on the back of the exam pages or in the extra pages provided at the back of the exam**. Simply providing numerical answers will only result in partial credit**, even if the answers are correct.

Draw a BOX or a CIRCLE around your answers to each problem.

Be sure to provide units where necessary.

GOOD LUCK!

PROBLEM	POINTS	MAX
1		20
2		20
3		15
4		25
5		20

Problem 1

20 points

a) In one or two sentences, why don't we build static CMOS with PMOS transistors in the pull-down network and NMOS transistors in the pull-up network? *(2 points)*

b) Any Boolean function can be implemented using multiplexers alone. *(2 points)*

True or **False**? (circle one)

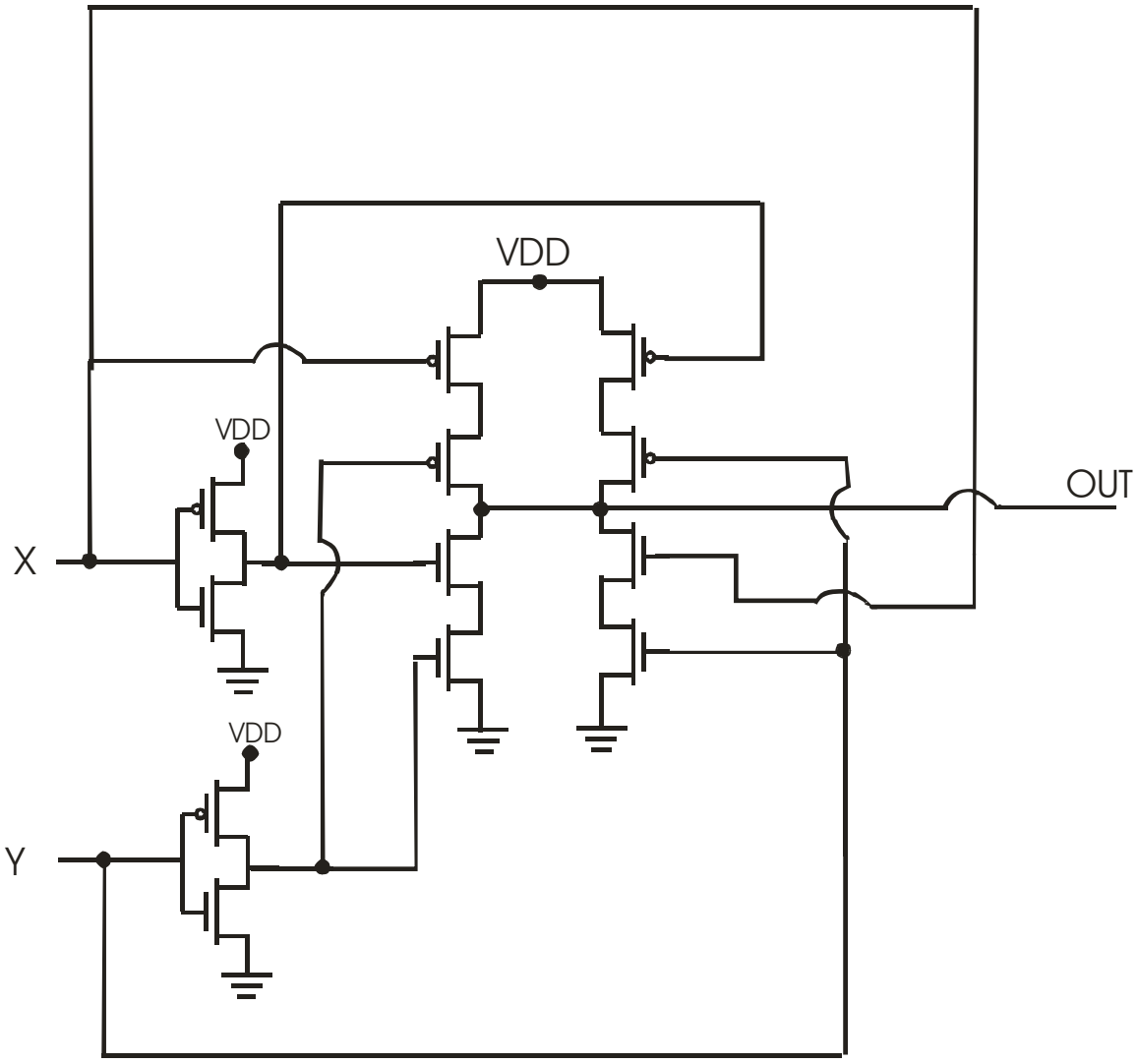
c) If you were to build a 64:1 multiplexer from 4:1 multiplexers, you would need how many 4:1 multiplexers? *(2 points)*

d) Fill in the following table with delay and cost as a function of n for each type of n-bit adder structure (use the "big O" notation). *(6 points)*

Adder type	delay	cost
Ripple carry		
Carry-Look-ahead		
Carry-Select		

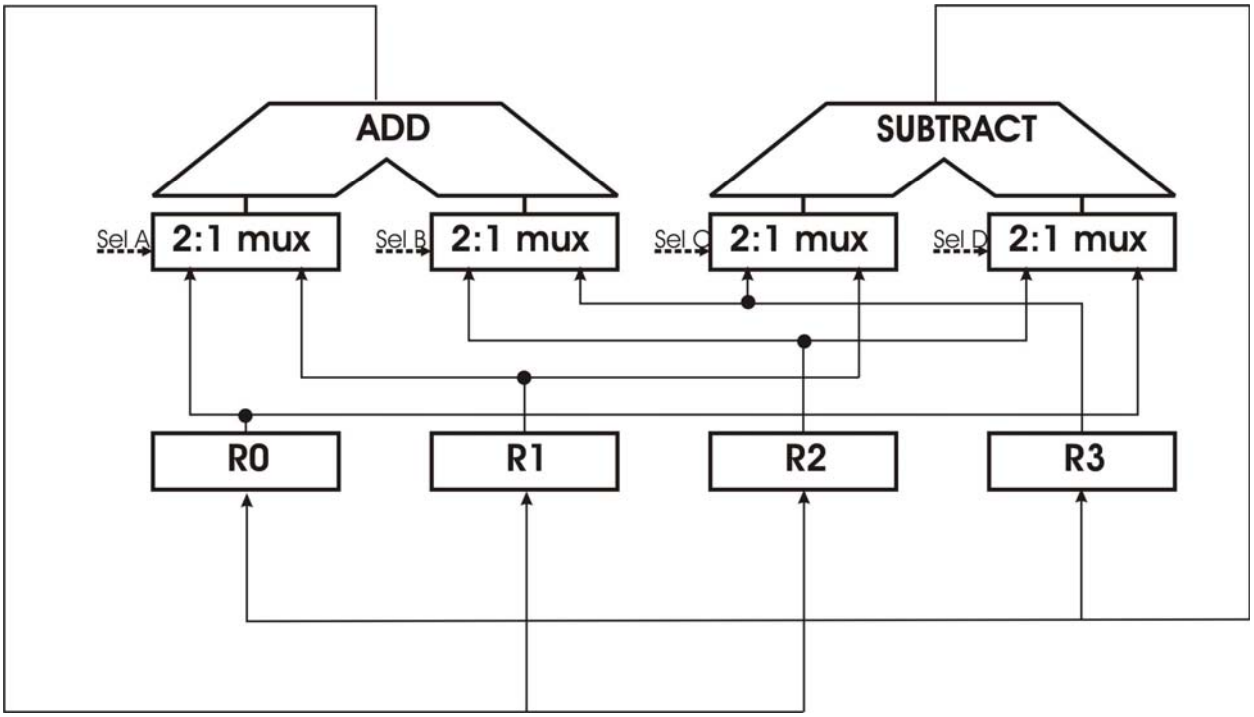
e) Draw the circuit-level diagram of an unclocked SR latch. *(3 points)*

f) What logic function does this circuit perform? A logic table **and** logic expression is required answer. *(5 points)*



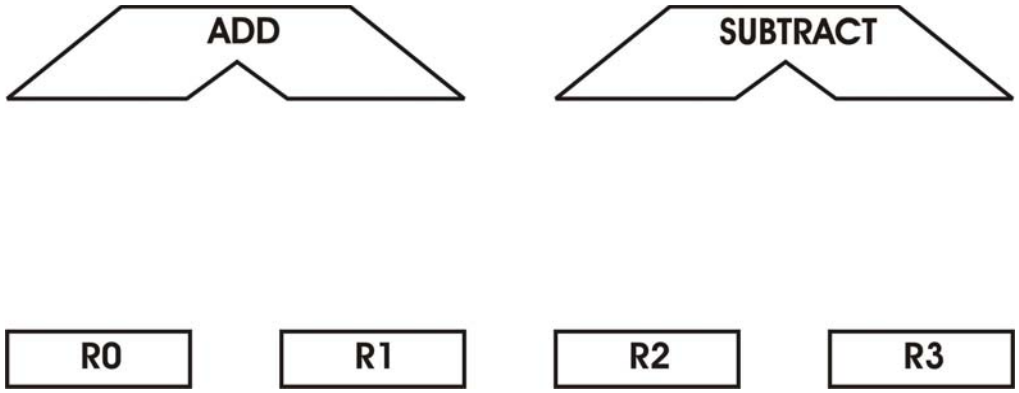
Problem 2 **20 points**

Consider the datapath below. All solid lines are 8 bit wide, the four dotted lines are 1 bit control lines. The functional unit at the right of the datapath is a subtractor; at the left is an adder. Assume you can only write to a single register per cycle. Two wires are shorted only if there is a black circle at the intersection.

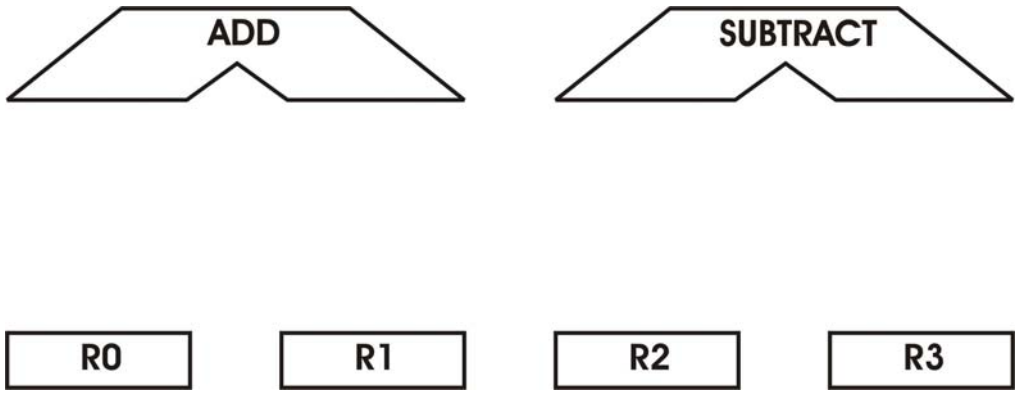


a) Write down ALL of the architectural level register transfer operations (e.g., Reg ← Reg op Reg) that can be executed in a single processor cycle. *(5 points)*

b) Using only the four registers and two functional units (ADD, SUB), design a bussing structure (ie. wires + muxes) for the datapath so you can implement ANY register-to-register ADD or register-to-register SUB (including the same register used as both sources and the destination). Both addition and subtraction should take place on the same cycle, but only one of those two results will be stored. Your bussing design must use the fewest possible additional wires to accomplish this task. Draw your bussing structure below. *(7.5 points)*



c) Revise your solution from b) for the case where ADD and SUB can occur simultaneously, but never have the same target register (it wouldn't make sense to write something into the same register from two different places at the same time!). Your design must use the fewest possible wires! *(7.5 points)*



Problem 3 **15 points**

The easiest way to perform arithmetic operations using the sign-and-magnitude system is to convert to 2's complement for your calculations. You are given the following components:

- A 4-bit universal shift register
- As many 1-bit full adders as you need
- As many 2-input XOR gates as you need
- As many 2-input NAND gates as you need

Design a circuit to convert a 4-bit sign-and-magnitude number to 2's complement number.

Problem 4 **25 points**

Back in the days of mainframe computing, there were two standard for textual information interchange – ASCII and EBCDIC. You are implementing a counter that counts in hexadecimal. The output of the counter is either in ASCII (0-9 = 48-57, A-F = 65-70) or EBCDIC (0-9 = 240-249, A-F = 193-198) depending on the value of a control switch (C = 0 → ASCII, C = 1 → EBCDIC). You are provided with the following parts:

- A 4-bit binary counter (Pins: CLK, RESET, C3-C0 outputs)
- A 4:16 DEMUX (Pins: G enable, S3-S1 select lines, Z output)
- A 32×8 bit ROM (Pins: A4-A0 address lines, D7-D0 data lines)
- As many NAND gates as you need

You may use some or all of the parts above. You may choose to program the ROM with whatever values you need.

a) Design the ASCII/EBCDIC counter. Use block diagrams for the individual parts and label the pinouts. *(15 points)*

b) Fill out the following table with the values that you chose to store in the ROM. You should write the data in decimal for convenience (and ease of grading). *(10 points)*

Address	Data	Address	Data	Address	Data	Address	Data
00000		01000		10000		11000	
00001		01001		10001		11001	
00010		01010		10010		11010	
00011		01011		10011		11011	
00100		01100		10100		11100	
00101		01101		10101		11101	
00110		01110		10110		11110	
00111		01111		10111		11111	

Problem 5 **20 points**

Your task is to design the control for a sequential multiplier. The two operands are called the *multiplier* and the *multiplicand*, and the result is the *product*. It is possible to implement a sequential multiplier using the successive addition method. This is illustrated with the example below for unsigned 4-bit magnitude operands and an unsigned 8-bit magnitude product (e.g., 3 times 4 is 12 in decimal):

Multiplier: 0011
Multiplicand: 0100
Product: 0000 1100

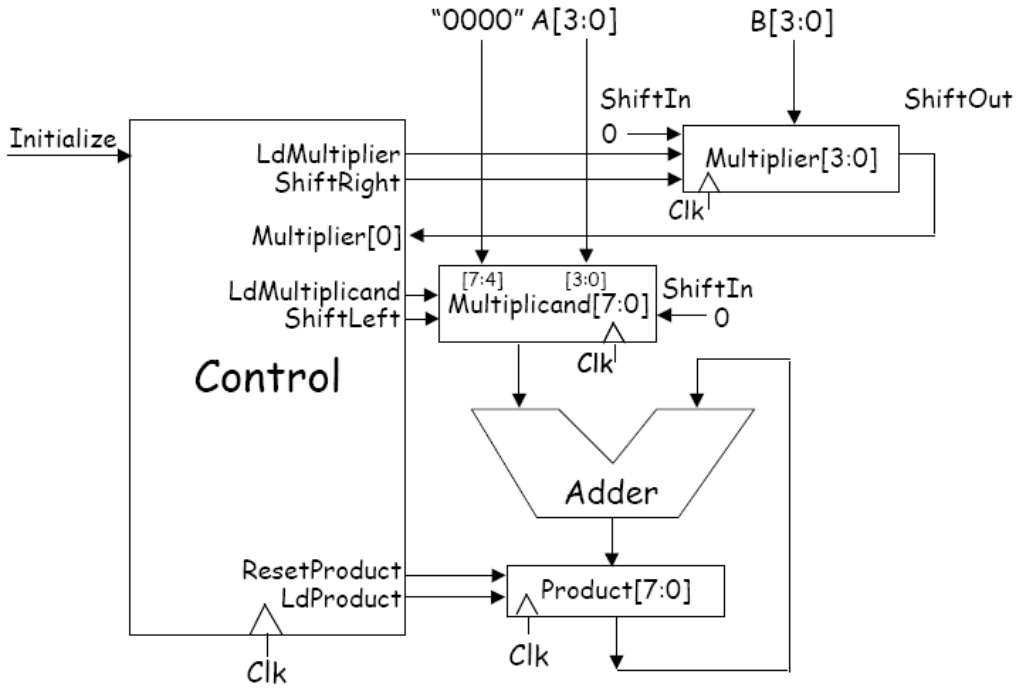
Start with the Product set to 0. The basic strategy is to examine the low order bit of the multiplier. If it is 1, then add the multiplicand to the running Product. Otherwise, don't do any addition. Shift the multiplier one position to the right, and the multiplicand one position to the left. This process repeats four times for a 4-bit Multiplier and Multiplicand. See the cycle-by-cycle results in the table below:

Cycle	Multiplier	Multiplicand	Product
Initialize	0011	0000 0100	0000 0000
Cycle 0, Multiplier[0]=1	0001	0000 1000	0000 0100
Cycle 1, Multiplier[0]=1	0000	0001 0000	0000 1100
Cycle 2, Multiplier[0]=0	0000	0010 0000	0000 1100
Cycle 3, Multiplier[0]=0	0000	0100 0000	0000 1100

High-level pseudocode for the multiplier is as follows:

```
Product = 0
For i = 0 to 3 do
      If Multiplier[0] = 1 then Product = Product + Multiplicand
      Shift right the Multiplier
      Shift left the Multiplicand
```

Given the following datapath, write the *verilog* description for a Moore Machine implementation of the multiplier control on the next page.



a) Write your Verilog below. When the Initialize signal is true, load A and B into the Multiplicand and the Multiplier, and set the Product to zero. When Initialize is no longer true, commence the computation of the product. *(10 points)*

b) Determine one way to accelerate the multiplication by taking advantage of special case values of the inputs. Briefly describe how you would change your control to take advantage of the special case you identified. *(10 points)*