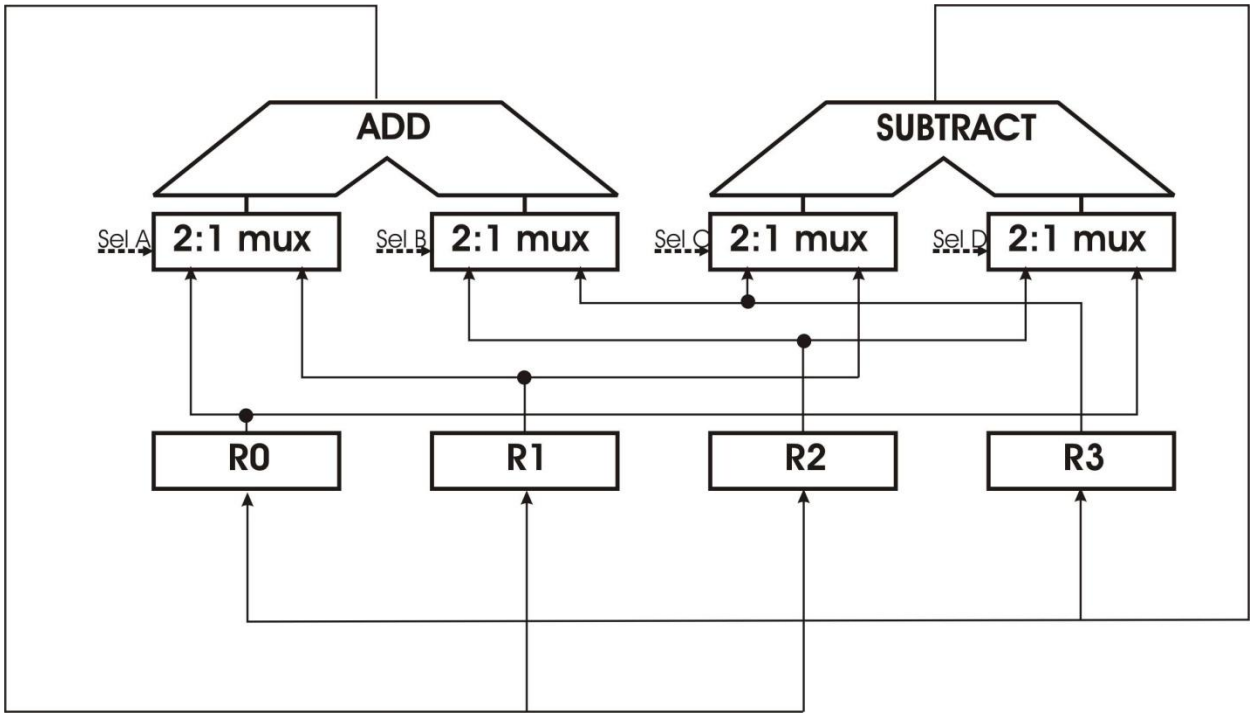
Problem 2 20 points

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)*

R0 <- R1 – R2
R0 <- R1 – R0
R0 <- R3 – R2
R0 <- R3 – R0

R3 <- R1 – R2
R3 <- R1 – R0
R3 <- R3 – R2
R3 <- R3 – R0

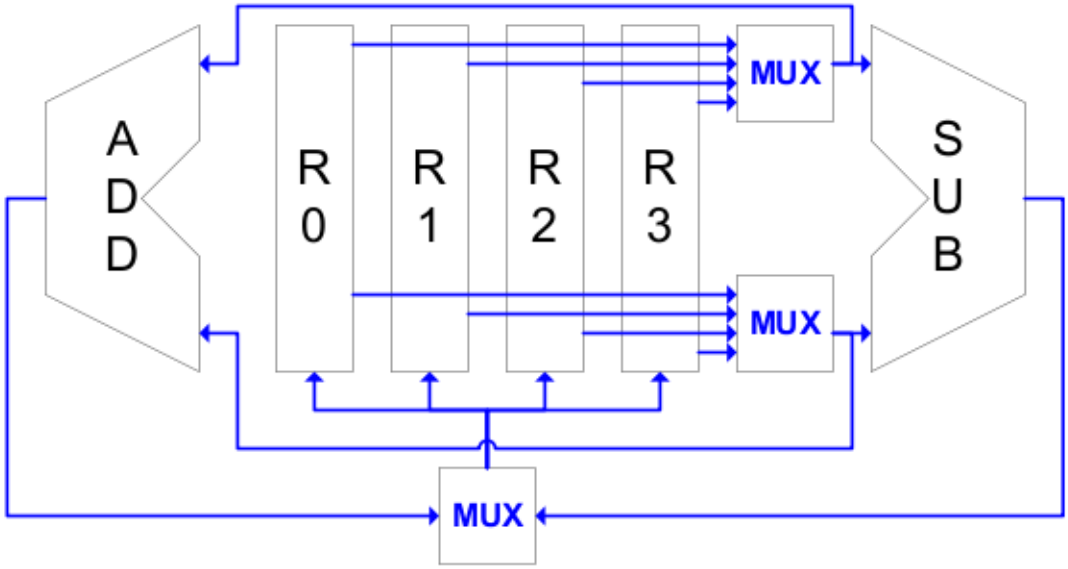
R1 <- R0 + R2
R1 <- R0 + R3
R1 <- R1 + R2
R1 <- R1 + R3

R2 <- R0 + R2
R2 <- R0 + R3
R2 <- R1 + R2
R2 <- R1 + R3

For this problem we were operating under the assumption that we would only ever write to a single register per cycle. Since this is a microprocessor like structure that is a reasonable assumption. We also assumed that each register had a separate write enable signal controlled by whatever (not shown) controller that controlled the muxs.

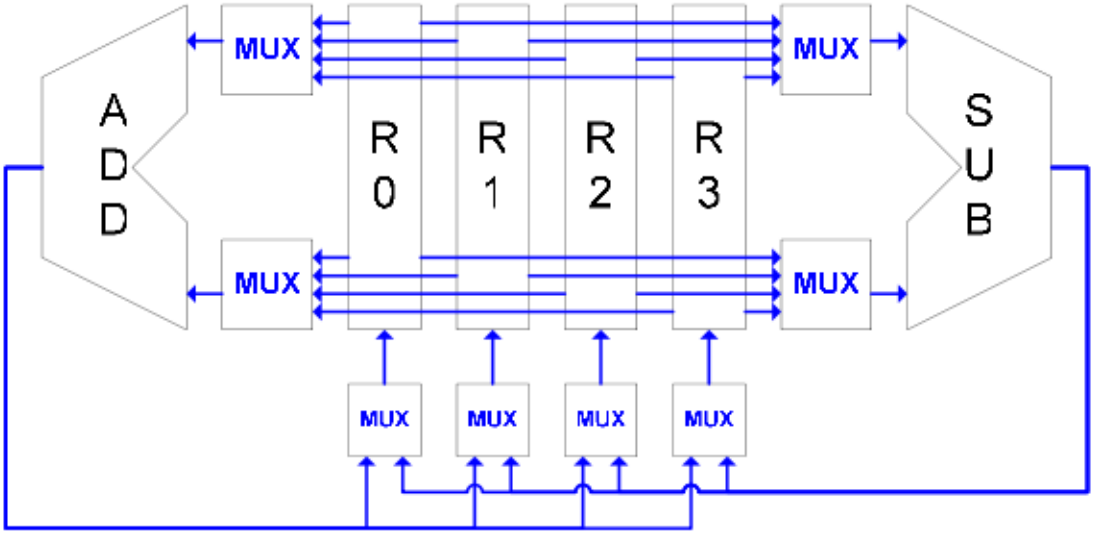
If you had a different set of assumptions, your answer may well be correct too, provided that you wrote down your assumptions.

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)*



In this problem, both addition and subtraction actually take place on the same cycle. However, only one of those two results would be used. The problem was loosely worded.

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 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

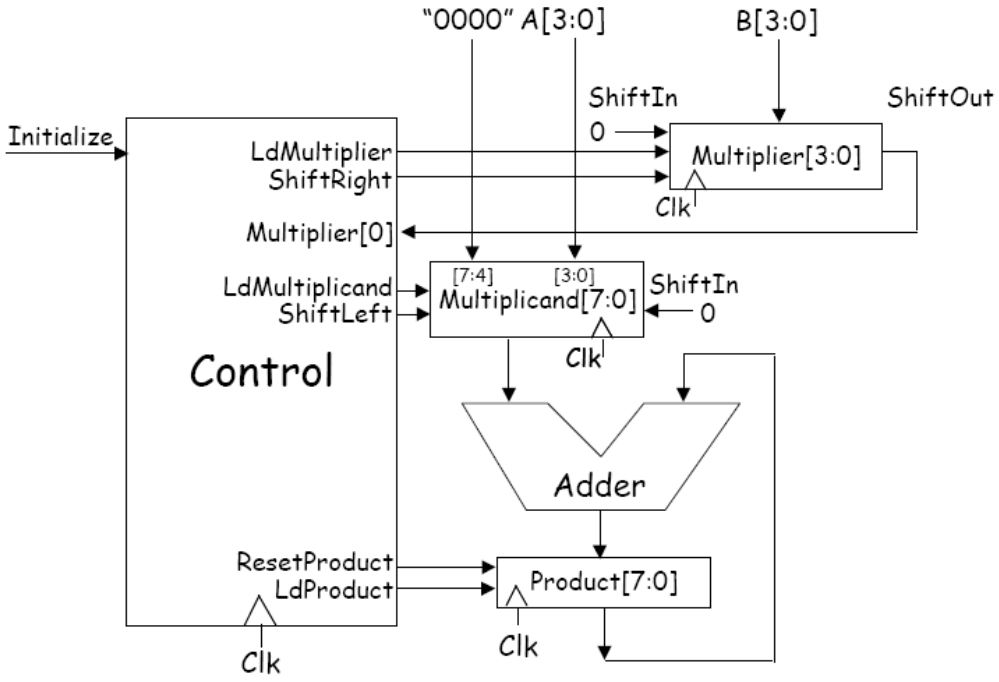
Start with the Product set to 0. The basic strategy is to examine the low order bit of the multiplier. If it is 1, then add the multiplicand to the running Product. Otherwise, don't do any addition. Shift the multiplier one position to the right, and the multiplicand one position to the left. This process 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)*

```
module    Control(Clock,    Initialize,    LdMultiplier,    ShiftRight,
                  Multiplier0, LdMultiplicand, ShiftLeft, ResetProduct,
                  LdProduct);
      input       Clock, Initialize, Multiplier0;
      output      LdMultiplier, ShiftRight, Multiplier0, ;
      output      LdMultiplicand, ShiftLeft, ResetProduct, LdProduct;

      wire        Enable;
      wire  [2:0] Count;
      reg         Start;

      Counter     #(3)  ACounter(   .Clock(     Clock),
                                    .Reset(     Start),
                                    .Count(     Count),
                                    .Enable(    Enable));

      always @ (posedge Clock) Start <=         Initialize;

      assign      Enable =                      ~Count[2];

      assign      LdMultiplier =                Start;
      assign      LdMultiplicand =              Start;
      assign      ResetProduct =                Start;

      assign      ShiftRight =                  Enable;
      assign      ShiftLeft =                   Enable;

      assign      LdProduct =                   Multiplier0;
endmodule
```

Of course there are an infinite number of answers to this question, and naturally we expect to see the normal two-always-block format answer on most of the midterms. In fact given that this is such a small counter (only 2 or 3 bits) you might not have used a counter at all, that's just fine too. Anything that works…

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)*

Any time the multiplier register is all 0s, the multiplication is over. Since none of the subsequent steps would actually involve an addition, the product register already contains the answer. A single 4-input NOR gate would save you a lot of cycles for some multiplications.

Other valid answers include multiplication by 0 (at the start), or multiplication by 1, both of which are easy to optimize.