6. In this question, we will examine if we can make the dimming operation from question 4 run faster if we design a specific instruction **dim** just for this purpose.

   (a) (5 points) Design a simple *combinational* circuit that takes a 32-bit input A and performs the dimming operation described in question 4. Complete the following Verilog code for this operation, or if you prefer, draw a detailed circuit schematic that explains the circuit.

```
module dim (  input  [31:0] A,
              output [31:0] Z);

  assign Z= {8'b0000_0000,
             1'b0,  A[23:17],
             2'b00, A[15:10],
             1'b0,  A[ 7:1]};

endmodule
```

   (b) (1 point) What is the propagation delay of this circuit? Assume all logic gates will have a propagation delay of 100 ps. *(Caution: tricky question.)*

> **Solution:** There are no active parts to this circuit, it contains only wiring and will virtually have no delay.

(c) (6 points) Now that we have the **dim** block designed, we want to enhance our processor with an instruction that uses this block. The *Control Unit* of MIPS has already been modified to generate a signal *ExecuteDim* that will be set to '1' whenever the new **dim** instruction has been decoded.

Two of your colleagues make two different suggestions.

- Josh says: "*Let us make **dim** an R-type instruction. The block **dim** can be placed in parallel with the ALU and it will take its input from SrcA. An additional multiplexer can select between the output of the ALU and the output of the **dim** block depending on the value of ExecuteDim. You execute **dim** by running* `dim $dest, $src`"
- Sandra says: "*We can combine the* `sw` *operation with **dim** and make it an I-type instruction. We can use the WriteData input of the RAM as an input to the **dim** block. A multiplexer can then select between the WriteData and the output of the **dim** block depending on the value of ExecuteDim. You execute **dim** by running* `dim $src, offset($addr)`*. The value in $src, will be dimmed and written to the address $addr+offset.*"

Does Josh's idea work? Does Sandra's idea work? Whose idea do you think is better? Why?

Solution: Both ideas would work. Sandra has a better idea, as it will further reduce the number of instructions by one.

```
# Josh version
loop:  lw   $a0, 0($s0)      # load one value
       dim  $a0, $a0         # execute the 'dim' function
       sw   $a0, 0($s0)      # dim store back value
       addi $s0, $s0, 4      # next pixel address
       beq  $s0, $s1, done   # end of loop?
       j    loop
```

```
# Sandra version
loop:  lw   $a0, 0($s0)      # load one value
       dim  $a0, 0($s0)      # dim and store back value
       addi $s0, $s0, 4      # next pixel address
       beq  $s0, $s1, done   # end of loop?
       j    loop
```

As long as you can make a reasonable case for it, answers defending the idea of Josh will also be accepted as correct. The correct answer also depends on the hardware implementation you came up with. In 6a/b, the expected answer only has wiring, therefore does not add much additional delay apart from a multiplexer. This is the reason why Sandra's idea works better. If the dim function had non-negligible delay, it could be argued that Josh's idea keeps this delay in parallel to the ALU and does not slow down the circuit.

(d) (6 points) Modify the standard single cycle block diagram given below so that it will allow the new **dim** instruction based on one of the ideas mentioned above. *Please explicitly state whose idea you implement.*