1. (20%)
   a. No additions to the datapath are required. A new row should be added to the truth table in Figure 5.20. The new control is similar to load word because we want to use the ALU to add the immediate to a register (and thus RegDst = 0, ALUSrc = 1, ALUOp = 00). The new control is also similar to an R-format instruction, because we want to write the result of the ALU into a register (and thus MemtoReg = 0, RegWrite = 1) and of course we aren’t branching or using memory (Branch = 0, MemRead = 0, MemWrite = 0).
   b. One possible solution is to add a new control signal called “Invzero” that selects whether Zero or inverted Zero is an input to the AND gate used for choosing what the new PC should be (thus a new multiplexor is introduced). The new control signal Invzero would be a don’t care whenever the control signal Branch is zero. Many other solutions are possible.

2. (20%)
   a. Looking at Figure 5.20, we see that MemtoReg and MemRead are identical except for sw and beq, for which MemtoReg is a don’t care. Thus, the modification will work for the single-cycle datapath. The modification will also work on the multiple-cycle datapath assuming that the finite state machine is changed so that MemRead is asserted whenever MemtoReg is.
   b. Using Figure 5.20, we discover that MemtoReg could be replaced by ALUSrc, RegDst could be replaced by ALUOp1, and either Branch or ALUOp0 could be replaced in favor of the other (their signals are identical). Note that in reality there would likely be additional rows present in the truth table to support other instructions, and it is quite likely that no control signals could be eliminated.

3. (30%)
   The key is to understand that it is the length of the longest path in the combinational logic that is determining the cycle time. Essentially, we compute the length of the longest path for each instruction and then must take the one with maximum value. At
present, the lw instruction is providing the longest path of length 8 ns.

a. The changes do not increase any paths beyond current maximums. Thus the cycle time is still 8 ns.

b. Consider the beq instruction. We now have a path which requires 10 ns to compute the branch address \((X + Y = 10 \text{ ns})\), and this is maximum. Thus, the cycle time increases to 10 ns.

c. You might be tempted to again conclude that the branch instruction will require \(X + Y = 9 \text{ ns}\) for branch address computation. This is not quite correct, however, because the second adder (requiring \(Y\) time) has two inputs, one of which is not available until after the instruction is read (the 16 immediate bits), and this takes 2 ns. Thus, the actual maximum length path is again 10 ns, and the cycle time is 10 ns.

4. (30%)
The rt field of the jump register instruction contains 0. So, along with rs we read $0. To get rs to the output of the ALU, we can perform a dummy addition of \(rs + $0\). We already have a path from ALUOut to the PC. We can use this path for loading the PC with rs. The finite state machine will need a new state for when the opcode is 000000 and the function code is 001000. The current condition for entering state 6 needs to be modified to make sure the function code is not 001000. In the new state, we want the appropriate combination of states 0 and 6: ALUSrcA = 1, ALUSrcB = 00, PCWrite, PCSource = 00, and of course we make sure an addition is performed with ALUOp = 00.