1. # CS 2734, Computer Organization II, Fall 2000
   # MIPS program giving answer to Final, question 1
   .globl main
   main:
   addu $s7, $zero, $ra
   # Start of answer to Question 1
   .data
   A:.space 40
   .text
   la $s0, A          # address of A
   addi $t0, $0, 0      # loop counter, start at 0
   addi $t1, $zero, 10  # to terminate loop
   addi $t2, $zero, 2   # value to store
   Loop:   sw $t2, 0($s0)     # store current $t0 into A
   addi $t0, $t0, 1     # increment loop counter
   addi $s0, $s0, 4     # increment pointer into A
   addi $t2, $t2, 2     # increment pointer into A
   bne $t0, $t1, Loop  # branch back to form loop
   # End of answer to Question 1
   # Print the array
   la $a0, A
   li $a1, 10
   jal Write_array
   jal Newl
   # Finish main
   addu $ra, $zero, $s7 # normal end of main
   jr $ra             # return to system
   # end of main

2. # CS 2734, Final Exam, Problem 2
   .globl main
   main:
   addu $s7, $zero, $ra
   # main for prob 2
   addi $a0, $0, 55     # Param1 = 55
   addi $a1, $0, 89     # Param2 = 89
   jal PrintSum        # call PrintSum
   # end of main for prob 2

3. Just the standard sw diagram for the single-cycle implementation.

4. This instruction is like the first part of lw or sw and the last part of add. No additional data lines or control lines are needed.
   Cycles 1 and 2 are common to all instructions and are the same.
   Cycles 3 and 4 are common to all instructions and are the same.
   Cycle 3 is the same as that for lw or sw:
   ALUOut = A + sign-extend(IR[15-0]); The control settings are the same as for those instructions: ALUSrcA = 1, ALUSrcB = 10 (2 decimal), ALUOp = 00.
   Cycle 4 (the last cycle) is almost the same as that of add:
   Reg[IR[20-16]] = AULOut. (add has Reg[IR[15-11]] = ALUOut)
   Thus we need RegWrite = 1, MemtoReg = 0, and RegDest = 0.
   (Note that add needs RegDest = 1, because the destination register is in bits 15-11 for add, while it is bits 20-16 for addi.)
5. Inputs E and F are 5-bit register numbers that are operand registers needed by the and instruction. (E for the upper input into the ALU, and F for the lower.) Input A is the 5-bit register number giving the destination register of the sub instruction. Input C is the same for the next instruction (just shown as “before” in the diagram). Input B is the signal to allow writing into the register file, so that writeback can occur. D is the same signal from the next pipeline stage (not used in this diagram).

One has A == E and B asserted, so the register value determined for register $2 by the sub instruction is the same as the first operand of the following and instruction, and B asserted means that writeback of $2 will occur.

The actual forwarding line is the dark line from the MEM stage, going down, over to the left, up and into the upper multiplexor and the lowest entry.

-------------------------------------------------------------------------

6. A stall is needed after the lw instruction. See Section 6.5. The Hazard Detection Unit notices that a lw instruction is in Stage 3 (by checking that the MemRead flag is 1). It also checks that the result of the lw is in a register needed by the next instruction. In this case it inserts a 1-cycle stall, by inserting zeros in for the control signals, and by deasserting the IF/IDWrite control line, so that nothing is written into the IF/ID on that cycle. It also deasserts the PCWrite signal, so that nothing is written into the PC. Thus the next instruction is _not_ written into IF/ID until IF/IDWrite is asserted again, when the flow of instructions can start up. Also the PC value is not updated until the next cycle.

Thus a “bubble” is created in the sequence of instructions going through. Two cycles later (see Fig. 6.48) when the lw instruction is in its final stage and when the following and instruction is in its execute stage, the value of $2 must be forwarded into the ALU for use by the and instruction, using the Forwarding unit as for other data hazards. At the same time, lw finishes loading the new value into $2.

-------------------------------------------------------------------------

7. (a) When the system starts your program, it starts at line 29. Then the “jal main” class the main function and starts the earlier code labeled with “main:”. Finally, control returns to line 30 which does a MIPS exit.

(b) Start in on line 17 when there is any exception. This will print the message “Duhh-hhhhh!”, fetch the EPC value (of the offending instruction, add 4 to it to give the next instruction, and return to that instruction.

-------------------------------------------------------------------------

8. When the clock signal is asserted (rising clock edge), the value of D (asserted) goes through the first D latch, but waits at the second D latch until the clock deasserts (falling clock edge). At this point the value of D gets all the way through the flip-flop