CS 2734, Org II, Spring 2002
Final Exam, Selected and Partial Answers

1.

```
    # CS 2734, Computer Organization II, Spring 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
            add     $t2, $0, $s0    # address of item in A
    Loop:   sw      $t0, 0($t2)     # store current $t0 into A
            addi    $t0, $t0, 1     # increment loop counter
            addi    $t2, $t2, 4     # 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
            jr      $ra
    ########## write an array ###############
    Write_array:
            addi    $sp, $sp, -4    # room for $ra on stack
            sw      $ra, 0($sp)     # save $ra because not leaf
        ## initialization for loop
            move    $s0, $a0        # $s0 = $a0 = start of A
            move    $s1, $a1        # $s1 = $a1 = N
            move    $t1, $zero      # start $t1 = 0, the index
    LoopA:  beq     $s1, $t1, EndA  # if (N == index) goto EndA
        ## write value for A[i]
            addu    $t2, $t1, $t1
            addu    $t2, $t2, $t2
            addu    $t2, $s0, $t2   # $t2 = index*4 + start of A
            li      $v0, 1
            lw      $a0, 0($t2)     # integer to print
            syscall
        ## write a blank
            jal     Blan
            addi    $t1, $t1, 1
            j       LoopA
    EndA:   lw      $ra, 0($sp)
            addi    $sp, $sp, 4
            jr      $ra
```

```
    ########## write newline ################
    Newl:   li      $v0, 4
            la      $a0, Newline
            syscall
            jr      $ra
    ########## write blank ##################
    Blan:   li      $v0, 4
            la      $a0, Blank
            syscall
            jr      $ra

            .data
    Blank:      .asciiz  " "
    Newline:   .asciiz  "\n"
    ############### output ##############################
    # four06% spim -file quiz4.s
    # 0 1 2 3 4 5 6 7 8 9
    ######################################################
```

Alternatively, the following code uses the mul pseudo-instr:

```
    ########## 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
    Loop:   mul     $t2, $t0, 4     # pseudo-instr, mult $t0 by 4
            add     $t3, $t2, $s0   # add to start addr of A
            sw      $t0, 0($t3)     # store current $t0 into A
            addi    $t0, $t0, 1     # increment loop counter
            bne     $t0, $t1, Loop  # branch back to form loop
    ########## End of answer to Question 1 ##################
```

2.

```
##### CS 2734, Final Exam, Problem 2 #######
        .globl main
main:   addu    $s7, $zero, $ra

########## MAIN FOR PROB 2 #################
        addi    $a0, $0, 12     # Param = 12
        jal     F               # call F
        add     $a0, $0, $v0    # $v0 = ret val
        li      $v0, 1          # print it
        syscall
        jal     Newl            # print newline

# Finish main
        addu    $ra, $zero, $s7 # normal end of main
        jr      $ra             # return to system
########## END OF MAIN ####################
```

```
########## PROB 2, function F ############
F:
        add     $v0, $a0, $a0
        jr      $ra
########## END OF FUNCTION F ##########

########## write newline ################
Newl:
        li      $v0, 4
        la      $a0, Newline
        syscall
        jr      $ra
########## DATE ##########################
        .data
Newline: .asciiz "\n"
#########################################
# Output:
#   24
#########################################
```

---

3. 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.

Fetching the instruction, updating the PC, and fetching
registers are the same as for all instructions.

The ALU does the same as for lw or sw:
ALUresult = (Output of Read data 1) + sign-extend(IR[15-0]);
The control settings are the same as for those instructions:
ALUSrc = 1, ALUOp = 00. (So that input to ALU becomes 010 (add).)

For the rest, we have to route the ALUresult around back into the
register file, using IR[20-16] for the register.
(Note that add has Reg[IR[15-11]] = ALUresult)
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.)

---

4. Just the standard lw diagram for the multi-cycle implementation.

---

5. In the upper diagram, register 2 is forwarded from the sub
and register 4 for the and instructions.
In the lower diagram, register 4 would be forwarded from the and,
except that register 4 is also the target of the or instruction,
so it is forwarded from there.

---

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) 14 bits for the index means 2^14 entries = 16K words = 64K bytes of data in the cache.
(b) The address should be a word address, so the low order 2 bits will always be 0s.
(c) These are the remaining bits of the address, 32 - 14 - 2 = 16. After we know that bits 1-0 are 0s, that bits 15-2 match because they are the same index entry, we need to check the remaining 16 bits 31-16 with the 16 bits in the tag field to see that the addresses are exactly the same.
(d) The hardware uses bits 15-2 as an index into the cache to directly access a word, with no searching. If bits 31-16 match the Tag field and if the Valid bit is on, there is a hit.
(c) See the four items at the bottom of page 551.

---

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

---