CS61C

Faux Midterm

17 November 2002

Problem	**Points**	**Score**
Problem 1: Boolean Logic	2	
Problem 2: Interrupts, Circular Buffer	6	
Problem 3: Sequential Logic and Verilog	9	
Problem 4: CPU Architecture	10	
Problem 5: Pipelining	2	
TOTAL	**29**	

Problem 1 (2 points, 5 minutes)

Consider the Boolean function represented by the following truth table.

input A	*input B*	*Output Q*
0	0	1
0	1	0
1	0	1
1	1	1

Provide a Boolean expression that relates the output Q to the input values A and B. Your expression should be immediately translatable to a circuit with at most three gates. The gates of the circuit should include only inverters and two-input AND and OR gates.

Problem 2 (6 points, 12 minutes)
At the end of this exam is the code for handling output (excerpted from our solution to project 3).

Part a

Suppose the project solution program is modified to print the string "Hello\n" prior to accepting any input. This results in print being called with a six-character string as argument. What are the contents of nextIn and nextOut when print returns (i.e. at the label alldone in the code)? Assume that in the time that the terminal takes to output a single character, the processor can execute thousands of instructions. Briefly explain your answer.

nextIn's value =

nextOut's value =

Part b

Now consider the unmodified project solution program. Again assume that in the time that the terminal takes to output a single character, the processor can execute thousands of instructions. The first call to print uses the argument

```
"(Input'x') Running Total =>  0x00000000 \X0D\n"
```

i.e. a 42-character string. What are the contents of nextIn and nextOut when print returns from this call? Briefly explain your answer.

nextIn's value =

nextOut's value =

Code from the project 3 solution

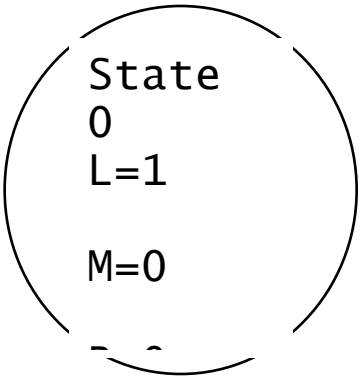
```
indone:                         # Next attend to output.
      lw $t0,nextOut # Is buffer empty? I.e. is nextOut equal to nextIn?
      lw $t1,nextIn
      bne $t0,$t1,notempty
      lui $t3,0xffff # If so, disable interrupts in the transmitter.
      sw $0,8($t3)
      j intDone

notempty:
      lui $t1,0xffff # Get base address of device registers.
      lw $t2,8($t1)  # Get status word for output.
      andi $t2,$t2,0x1  # Mask out all but ready bit.
      beq $t2,$0,intDone    # Return if not ready.
      la $t2,buffer  # Get buffer base address.
      addu $t3,$t2,$t0  # Add offset.
      lb $t2,0($t3)  # Get the character from the buffer.
      sb $t2,12($t1) # Output character to terminal.
      addiu $t1,$t0,-1  # Decrement index.
      andi $t1,$t1,31   # Wrap around from –1 back to 31.
      sw $t1,nextOut

intDone:
      ...
buffer:  .space 32# Characters waiting to be output.
nextIn:  .word 0  # Index of next place to insert character into buffer.
nextOut: .word 0  # Index of next character to remove from buffer.

print:
      lb $t3,0($a0)  # Fetch next character to store in buffer.
      addiu $a0,$a0,1   # Check for end of string.
      beq $t3,$0,alldone
      lw $t0,nextIn  # Fetch current input index.
      addiu $t1,$t0,-1  # Compute new input index after we store the next character.
      andi $t1,$t1,31   # Wrap around from –1 back to 31.

chkfull:
      lw $t2,nextOut # Is buffer full? I.e. is nextIn just before nextOut?
      beq $t2,$t1,chkfull  # If so, just keep checking until things get better.
      la $t4,buffer
      add $t4,$t4,$t0
      sb $t3,0($t4)  # There is space in the buffer; store character.
      sw $t1,nextIn  # Update "nextIn" index.
      lui $t3,0xffff # Make sure interrupts are enabled in the transmitter.
      addiu $t2,$0,2
      sw $t2,8($t3)
      j print  # Go back for more characters.

alldone: jr $31   # Return to caller.
```

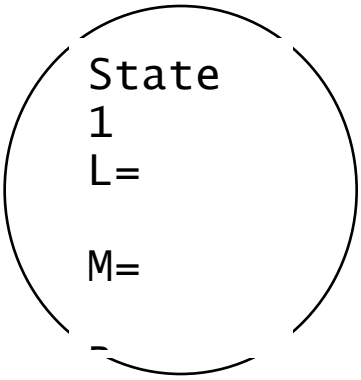
Problem 3 (9 points, 25 minutes)

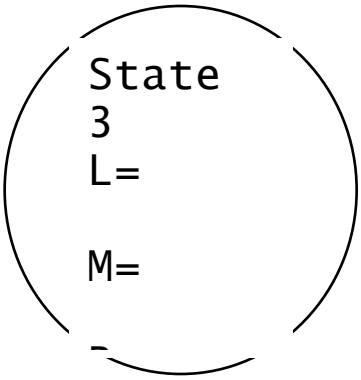
A friend would like you to build an "electronic eye" for use as a fake security device. The device consists of three lights lined up in a row, controlled by the outputs left (L), middle(M), and right(R), which, if asserted, indicate that the corresponding light should be on. Only one light is on at a time, and the light "moves" from left to right and then from right to left, thus scaring away thieves who believe that the device is monitoring their activity.

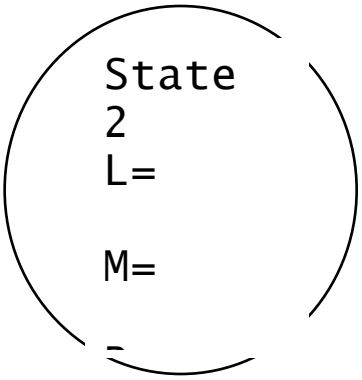
Part a

Assume the system has 4 states from 0 to 3 that it traverses in order. Fill in the state diagram below. Be sure to include the outputs that correspond to each state. Assume that transitions between states occur on every clock pulse and that state 0 corresponds to the state with the left light on. Don't worry about what state the device starts in.



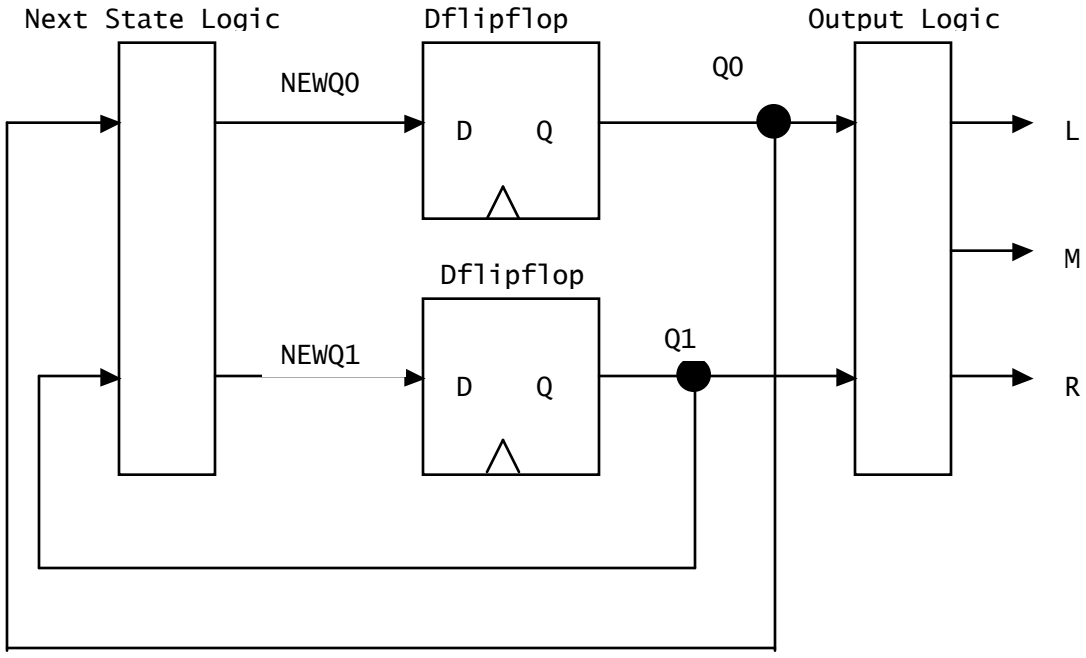






Part b

The figure below is a block diagram of one possible implementation of the eletronic eye. The two D flip flops hold the current state value. Your job is to fill in the truth table and write logic equations for the blocks labeled "Next State Logic" and "Output Logic" (Note that the rate of the eye's movement will be controlled by the clock speed.)



Truth Table:

Q0	Q1	NEWQ0	NEWQ1	L	M	R
0	0					
0	1					
1	0					
1	1					

Logic Equations:

NEWQ0 =

NEWQ1 =

L =

M =

R =

Part c

Using the module definitions provided below, fill in the necessary Verilog code to implement the modules.

```
module outputLogic( left, middle, right, Q );




endmodule // outputLogic

module nextStateLogic( nextQ, Q );




endmodule // nextStateLogic
```

Problem 4 (10 points, 30 minutes)

Consider the addition of the max instruction to the MIPS instruction set:

```
max   Rdest, Rsource1, Rsource2
```

It stores the larger of the values in registers Rsource1 and Rsource2 into register Rdest.

Part a

Design a machine representation for the max instruction that's consistent with the existing MIPS instructions. Clearly indicate the purpose of each bit field in the instruction, using the format of Patterson and Hennessy appendix A. A table of op codes appears at the end of this exam for reference. (However, it is not very clear so you should probably refer to your book.)

Part b

The next page contains a copy of Figure 5.19 from Patterson and Hennessy. Indicate by descriptions below and by additions to Figure 5.19 what changes to the datapath are necessary and what values existing signals must take on to implement the max instruction. Briefly explain your answers. You may assume that a new Max signal is provided by the instruction decoder. Your changes should *not* involve changing the ALU or adding a new ALU.

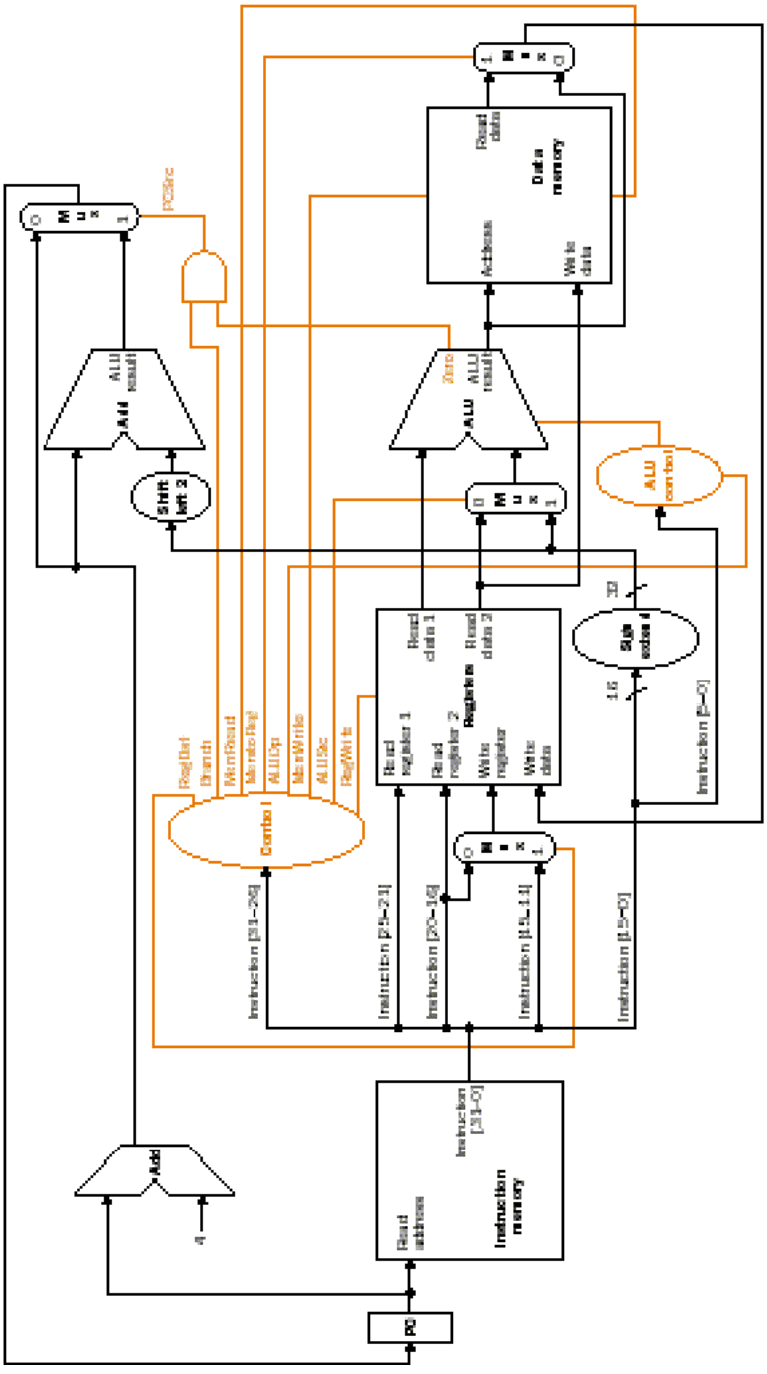
Values of existing signals:

```
Max = 1                  Branch =
MemRead =                MemtoReg =              MemWrite =
RegDst =                 RegWrite =
ALUSrc =                 ALUOp =
ALU control =
```

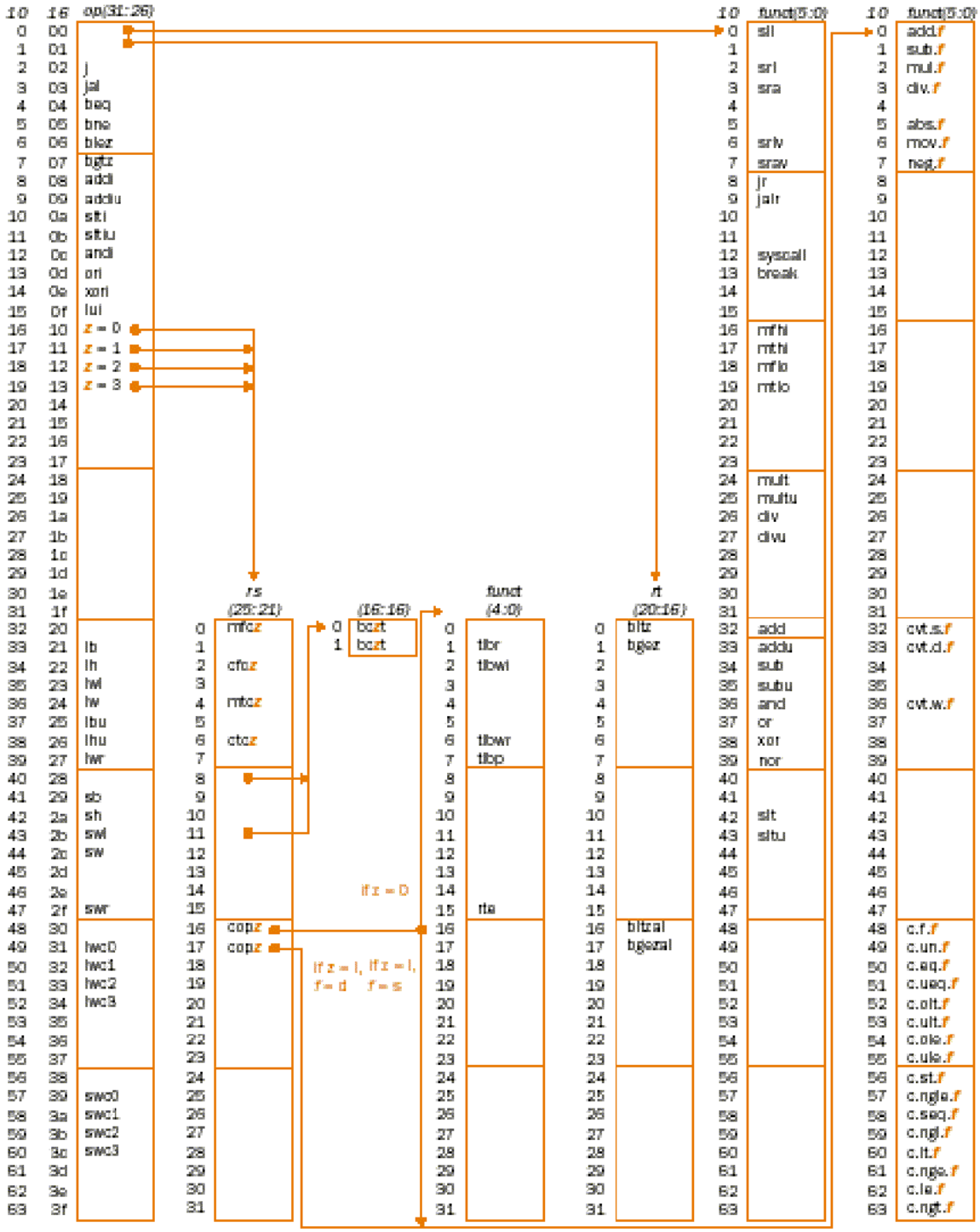
Brief explanation of control signal values:

Other changes:

Patterson and Hennessy Figure 5.19



Patterson and Hennessy Figure A.19



Problem 5 (2 points, 8 minutes)

Consider the following function, which returns the sum of all the elements in its argument, a linked list of four-byte integers. This was written to be run in spim, which normally does not simulate pipeline-related features such as branch and load delays.

```
sum:
   move  $v0,$0          # start the running total at 0

loop:

   beq   $a0,$0,return

   lw $t0,0($a0)         # get the element in the current node

   add   $v0,$v0,$t0     # add it to the running total

   lw $a0,4($a0)         # get the pointer to the next node

   j  loop

return:  jr $ra
```

Add no-op instructions to the above code to make all stalls and delayed branches explicit. Assume all forwarding features described in Patterson and Hennessy section 6.1 have been provided.