CS 61C Spring 2015
Guerrilla Section 1: Hardware & CPU

Problem 1:

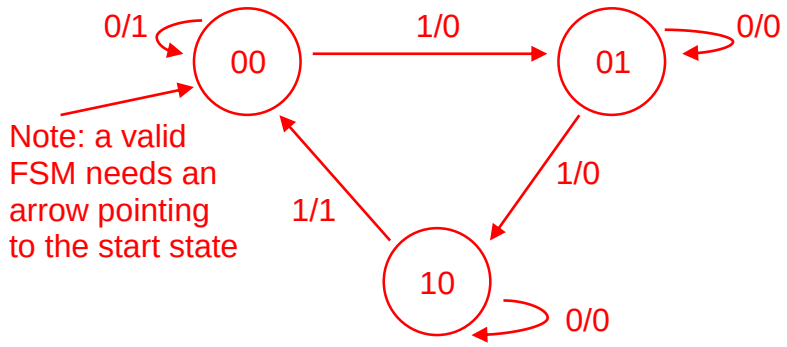
a) Convert the following truth table to a Boolean expression and simplify it. An X means we don't care about the value of that output (it can be either 0 or 1).

A	B	C	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	X
1	1	1	X

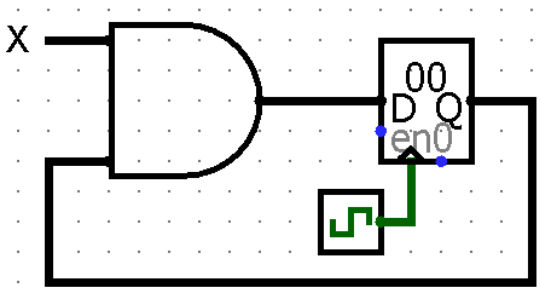
The trick is to set the output of the last 2 rows to 1.
Then we get:
!ABC + A!B!C + AB!C + ABC
So we can group terms 1 and 4 and terms 2 and 3:
=BC(!A + A) + A!C(!B + B)
=BC + A!C
This solution uses 4 gates (2 AND, 1 OR, 1 NOT).

b) Draw the transition state diagram from a FSM that reads a binary string bit-by-bit and outputs whether the total number of 1s seen since the beginning is divisible by 3.

State = (number of 1s seen) % 3



c) For the circuit below, assume that the setup time is 15ns, hold time is 30ns, and the AND gate delay is 10ns. If the clock rate is 10 MHz and x updates 25ns after the rising edge of the clock, what are the minimum and maximum values for the clk-to-Q delay to ensure proper functionality?



Min: _____20_____ns

Max: _____75_____ns

If the clk-to-Q delay is too fast, the input to the register will change before the hold time is finished. Thus, the minimum clk-to-Q delay is $t_{hold} - t_{AND} = 30 - 10 = 20$ns.

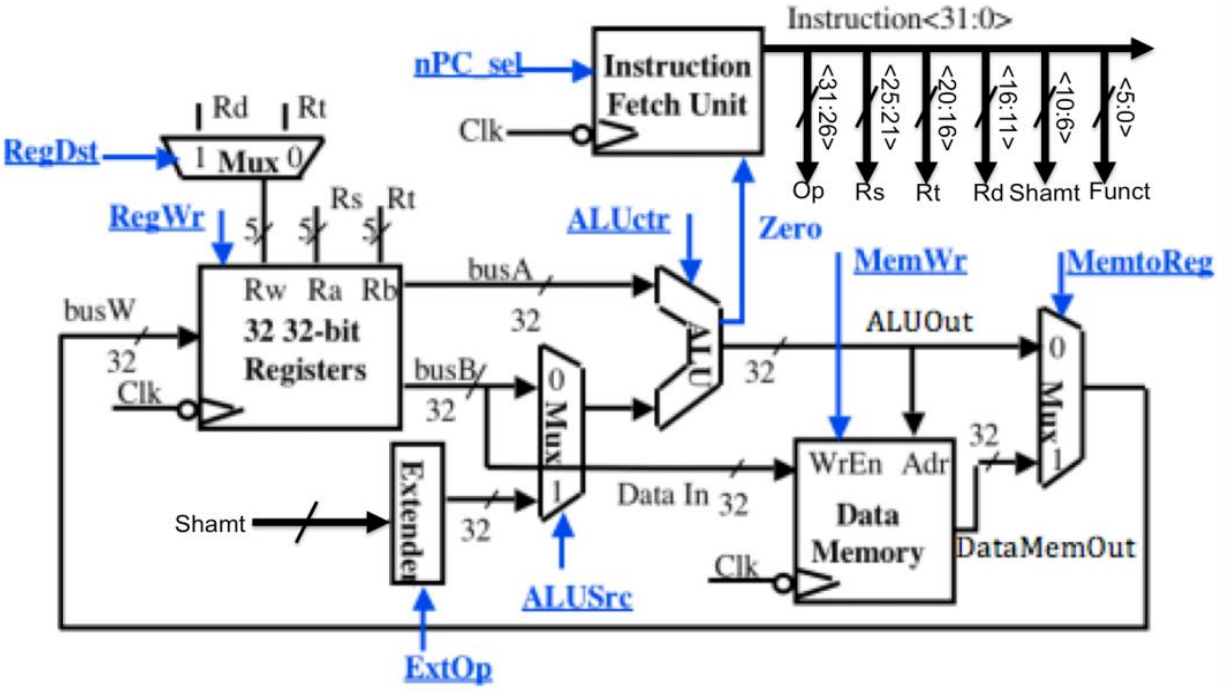
On the other hand, we must make sure the critical path is no longer than the clock period, which is 100 ns (= 1/(10 MHz)). In other words, $t_{setup} + t_{AND} + t_{clk\text{-}to\text{-}Q} \leq 100$ns, or $t_{clk\text{-}to\text{-}Q} \leq 100\text{ns} - t_{setup} - t_{AND}$. Solving yields $t_{clk\text{-}to\text{-}Q} \leq 75$ ns.

Problem 2 (adapted from Sp04 Final):

We want to implement a new I-type instruction `swai` (store word then auto-increment). The operation performs the regular `sw` operation, then increments the value in the `rs` register by 1.

The RTL for the `swai` instruction is:

```
Mem[R[rs] + SignExtImm] = R[rt]; R[rs] = R[rs] + 1; PC = PC + 4
```



a) Modify the single-cycle MIPS datapath (shown above), and describe your changes in the space below. Your modification may use simple adders, mux chips, wires, and new control signals. You may replace original labels where necessary.

1. Add a new adder whose inputs are R[rs] (busA) and a constant 1. I'll label the output as RsPlusOne
2. Either connect RsPlusOne to the MemToReg mux, making it a 4-to-1 mux, and make MemToReg a two-bit signal OR mux RsPlusOne with busW with a new mux (and introduce a new control signal)
3. Either feed rs into the RegDst mux, making it a 4-to-1 mux, OR mux the output of the RegDst mux with rs using a new mux (and introduce a new control signal)

b) Fill out the values of the control signals in the table below, including any new control signals that you have added in part A.

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg		
rs	1	PC+4	sign	1	add	1	RsPlusOne		

Answers to part B depend on changes made in part A

Problem 3 (adapted from Su13 Final):

Assume that we run the following snippet of code on a 5-stage pipelined MIPS CPU with **no optimizations**. Branch checking is done in the execute stage. Assume that `$a1 = 1` at the beginning of the code.

```
            lw $t0, 0($a0)
loop:       beq $a1, $0, exit
            sll $t0, $t0, 2
            addiu $a1, $a1, -1
            sw $t0, 0($a0)
            j loop
exit:
# when we reach the exit label, we're done
```

a) After which instructions are stalls needed? What is the total number of clock cycles for these instructions to finish execution (when the pipeline becomes empty)? You may use the table below as scratch space.

There is a data hazard between the lw – beq instructions (1 stall), a control hazard after beq (2 stalls), a data hazard between sll – sw (1 stall), and a control hazard after j (1 stall). However, since we put two stalls after the beq, the lw – beq stall is already accounted for. Thus, there are 4 stalls in total.

Total number of cycles = 15

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw	IF	ID	EX	M	WB											
beq		IF	ID	EX	M	WB										
sll					IF	ID	EX	M	WB							
addiu						IF	ID	EX	M	WB						
sw								IF	ID	EX	M	WB				
j									IF	ID	EX	M	WB			
beq											IF	ID	EX	M	WB	

b) Consider the following optimizations *separately*. How many FEWER cycles are taken for the addition of each optimization?

a. Forwarding
   1 cycle, only gets rid of data hazard from sll - sw

b. Branch prediction of never take branch
   1 cycle, gets rid of control hazard from beq – sll, but not the data hazard from lw-sll