Name: (as it would appear on official course roster)		
UCSB email address:	**@ucsb.edu**	**Perm ID Number:**
Lab Section Time:		
Optional: name you wish to be called if different from above		
Optional: name of "homework buddy" (leaving this blank signifies "I worked alone")		

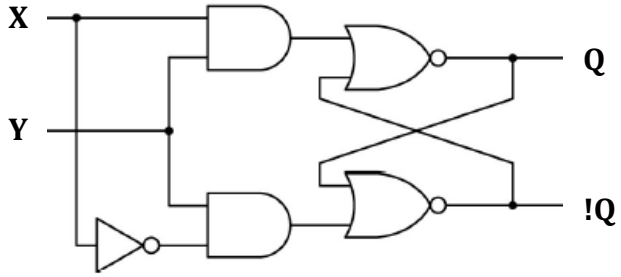
Lab 08: Combinatorial and Sequential Logic

Assigned: *Thursday, March 5th, 2020*
Due: *Tuesday, March 10th, 2020*
Points: *120 (normalized to 100)*

- You may collaborate on this homework with AT MOST one person, an optional "homework buddy".
- MAY ONLY BE TURNED ON **GRADESCOPE as a PDF file**.
- There is NO MAKEUP for missed assignments.
- We are strict about enforcing the LATE POLICY for all assignments (see syllabus).

IMPORTANT! Read the questions CAREFULLY before answering them! Some of the questions here will end up with you making a drawing – **make these drawings clear and readable!**

1. (5 pts) Consider this digital circuit:

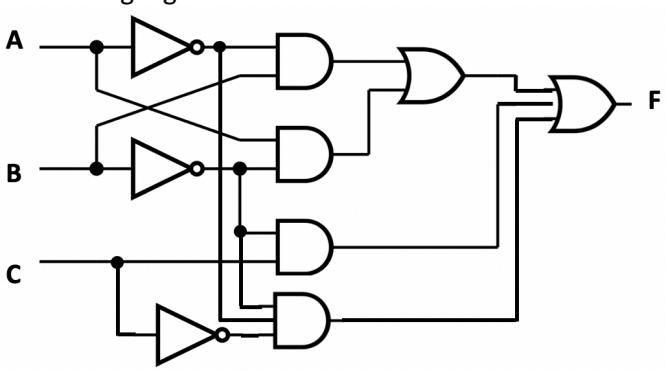


   a) (1 pts) What is the name of the sequential logic circuit (group of logic gates) that is directly "driving" the outputs Q and !Q?

   b) (3 pts) Write out the truth table for the full circuit.

   c) (1 pts) Describe (using words! ☺) what this circuit does.

2. (20 pts) Given the following digital circuit:



a) (10 pts) Construct the truth table for this circuit.

b) (10 pts) Using your answer from part (a), write the ***optimized*** logic function that describes the diagram in the form of "sum-of-product". You can use Karnaugh maps (if so, show optimal groupings!) or other techniques, if you like.

3. (30 pts) Design and draw an instruction decoder (not MIPS, just our own custom CPU!). This is combinatorial logic that takes in an "**op-code**" in 5 bits and outputs an ALU-control, "**ALUC**", in 3 bits. Your design **has to be optimal.**

   Assume that you only have the following op-codes and their related ALUC outputs:

Instruction type	**op-code[4:0]**	**ALUC[2:0]**
Add	00000	111
Sub	00001	110
AND	00010	101
NOR	00011	100
XOR	00100	000

   The best way to complete the required design is to determine how every bit of the ALUC outputs relates to each of the op-code inputs. This requires you to use a truth table and K-map approach. **Hint**: Take advantage of the fact that op-code bits 4 and 3 are always 0 (it makes the K-Maps easier to do).

   Ultimately, I want to see (a) a truth table, (b) multiple K-maps, and (c) a drawing of the logic design using only AND, OR, and NOT gates. You have these logic gates in unlimited amounts. Write your solutions on this and the next page.

(this space for Question 3)

4. (30 pts) Design and draw a simple single-bit ALU, which is partially based on the decoder exercise (Question 3). This ALU takes the following inputs:

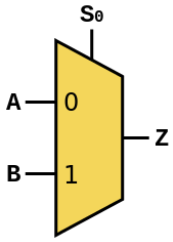
Input Name	Input Description
Operation	Which operation is to be performed, specified with a select input (**ALUC**). There are three possible operations: 1. AND, specified when ALUC = 101 2. NOR, specified when ALUC = 100 3. XOR, specified when ALUC = 000 *(yes, we are ignoring the "add" and "sub" operations for this exercise)*
Data_A	The first operand, specified with a single bit
Data_B	The second operand, specified with a single bit

The ALU returns the following single output:

Output Name	Output Description
ALUout	The result of whatever operator was chosen, applied to the given operands. For example, if NOR was chosen, then the result should be !(Data_A | Data_B)

To complete this task, you may use **only** the following components, in unlimited supply:

- AND, OR, and NOT gates, using the notation shown in lecture.
- 2-input multiplexers, which take a selector bit S0 and two single-bit input operands A and B, and return a single-bit output Z. They should be drawn using the symbol below (again, as used in lecture).



Write your solution on this and the next page.

(this space for Question 4)

5. (30 pts) Design and draw a a small register file (i.e. set of registers). The register file contains two registers, each one bit long, which are named **regA** and **regB**, respectively. The register file allows two registers to be read simultaneously, and also allows a register to be written to. The register file takes the following inputs:

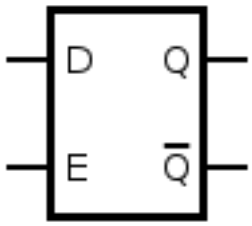
Input Name	Input Description
R0	The first register to read, as a single bit. If 0, then **regA** should be read. If 1, then **regB** should be read.
R1	The second register to read, as a single bit. If 0, then **regA** should be read. If 1, then **regB** should be read.
WR	Short for "Write Register". Specifies which register to write to. If 0, then **regA** should be written to. If 1, then **regB** should be written to.
W	The data that should be written to the register specified by **WR**. This corresponds to a single bit.
WE	Short for "Write Enable". If 1, then we will write to a register. If 0, then we will **not** write to a register. Note that if **WE** = 0, then the inputs to **WR** and **W** are effectively ignored.

The register file returns the following two outputs:

Output Name	Output Description
O1	Value of the first register read. As described previously, this depends on which register was selected to be read, via **R0**.
O2	Value of the second register read. As described previously, this depends on which register was selected to be read, via **R1**.

To complete this task, you may use **only** the following components, in unlimited supply:

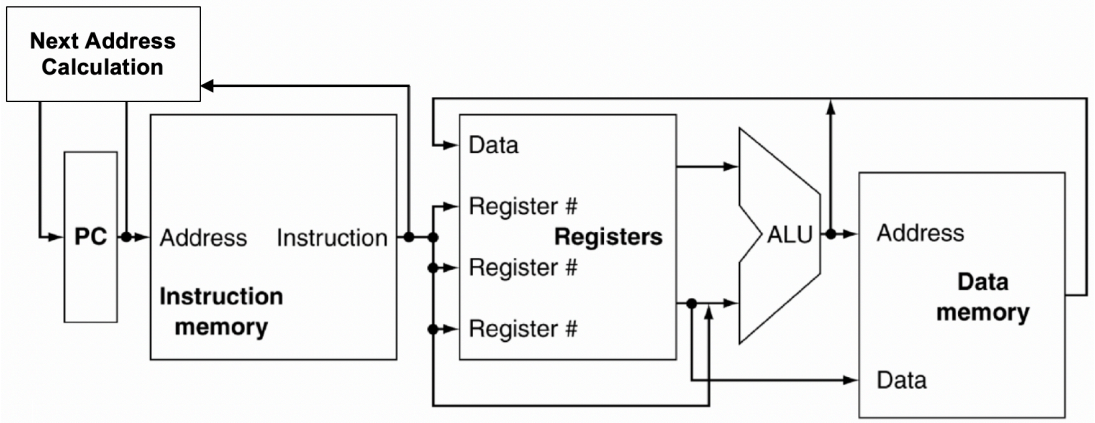
- AND, OR, and NOT gates, as well as 2-input multiplexers, using the notation shown in lecture.
- Gated D-latches, which take a single-bit input D and an "enable" bit E, which is used to indicate that the value of D should be saved within. They produce single-bit outputs Q and !Q, which represent the value stored in the latch along with its negation, respectively. They should be drawn using the symbol below (as used in lecture):



Write your solution on the next page.

(this space for Question 5)

6. (5 pts) This is a ***simplified*** diagram of a MIPS CPU, showing various functional logic/hardware blocks. **PC** is the Program Counter register, Instruction memory is the part of memory where the program resides (aka Text section of memory), **Next Address Calculation** is the part of the CPU that decides what the next value in **PC** should be, **Data memory** is the part of memory where most data is stored (aka dynamic and static memory), **ALU** is the arithmetic logic unit, and **Registers** is the register bank where all 32 registers of MIPS reside.



Knowing what we know about the functions of these blocks, answer the following questions:

a. (1 pt) Name the blocks that would *directly* involve the instruction **`sub $t1, $t2, $t3`**?

b. (1 pt) Name the blocks that would *directly* involve the instruction **`addi $t1, $t2, 64`**?

c. (1 pt) Name the blocks that would *directly* involve the instruction **`beq $t1, $t2, loop`**?

d. (1 pt) Name the blocks that would *directly* involve the instruction **`lw $t1, 0($sp)`**?

e. (1 pt) Name the blocks that would *directly* involve the instruction **`sw $t1, 4($t0)`**?