CIS 240 Fall 2021: Midterm

Oct 25, 2021

Name :

____

Please write your name on the exam. You can answer the questions on this exam sheet in the space provided. Please put your initials at the top of each page in case the pages become separated.

Please read the following pledge and sign in the space below:

I neither cheated myself nor helped anyone cheat on this exam

Sign Here :

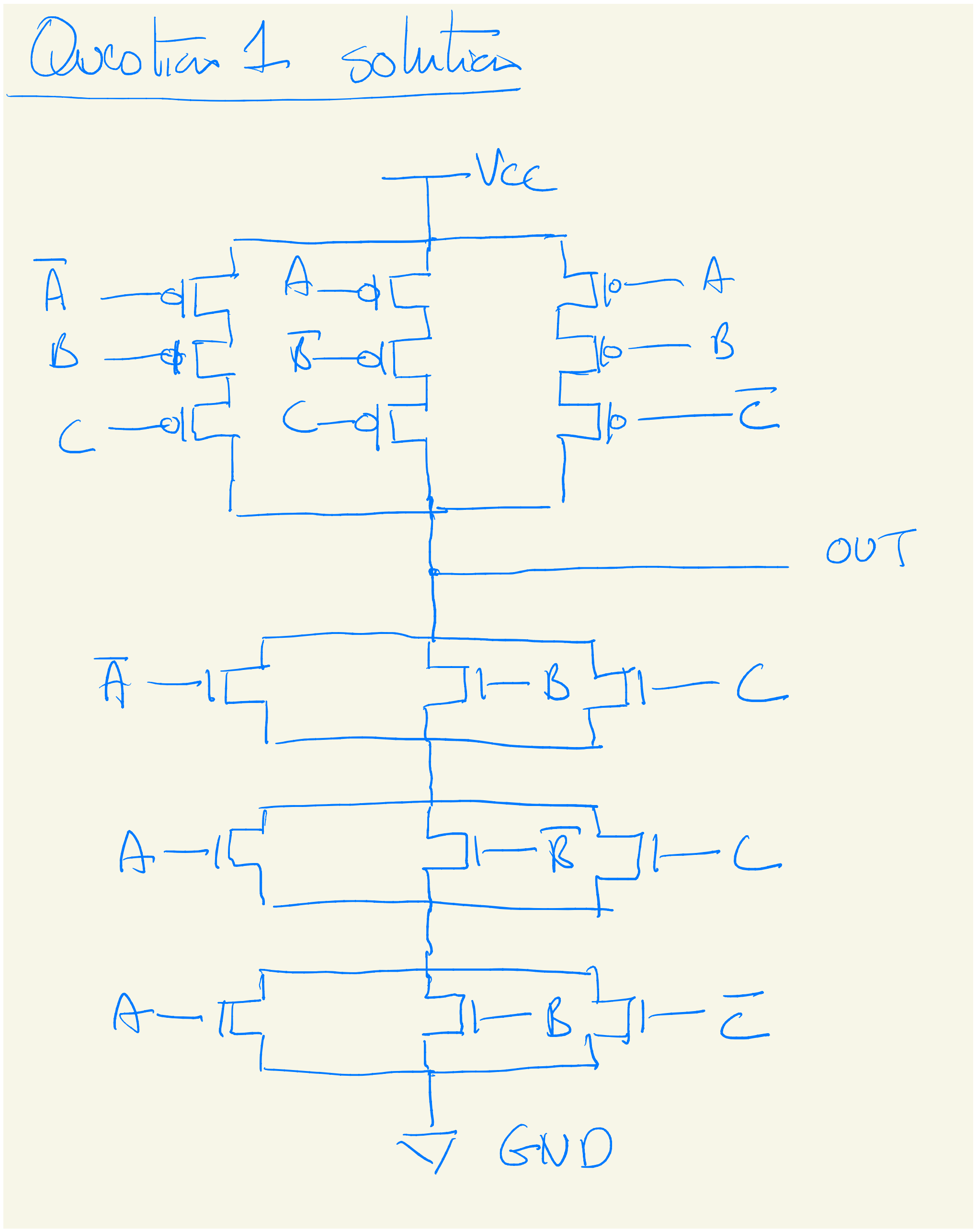
Do not write in the table below:

1	**2.1**	**2.2**	**2.3**	**3**	**4.1**	**4.2**	**5**

Question 1 {10 pts}

Your job is to design a proper CMOS circuit which has 3 inputs, A, B and C and where the output is 1 if and only if exactly one of these 3 inputs is high. You can assume that you also have access to negated versions of all the input signals. Your solution must be a single CMOS circuit consisting of complementary pull up and pull down transistor networks. Please label the inputs and outputs of your circuit clearly on your schematic.

Answer:



Note that this is just one of many possible answers.

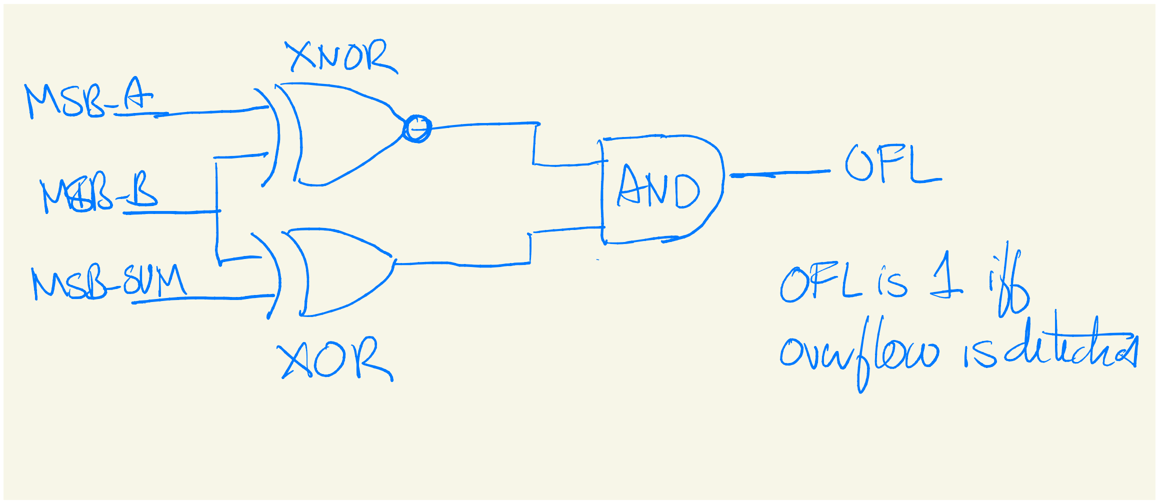
Question 2

In class we discussed the fact that adding 2C numbers can result in arithmetic overflow which can cause the final result to be horrendously incorrect. Some computers try to improve on this state of affairs by implementing **saturating arithmetic**. When the result of an addition would be larger than the largest positive n-bit 2C value the saturating adder would produce this maximum n-bit 2C value, similarly if the result would be more negative than the most negative n-bit 2C value the saturating adder produces the most negative n-bit 2C value. The result of the addition still isn't correct in these cases, but it's the closest answer that could be produced given the constraints of the n-bit 2C representation. In this question you will design a 5-bit saturating adder. Please help the grader by clearly labeling the inputs and outputs of each of the circuits you produce. Feel free to add comments to explain your designs.

Part 1 {10 pts}: Produce a gate level circuit that produces a 1 if and only if the addition of two 5-bit 2C numbers, A and B, would result in an overflow situation. That is a situation where the correct result would be outside the range of numbers that can be represented in 5-bit 2C format. Your circuit can use any of the following signals as inputs: MSB_A – Most significant Bit of input number A, MSB_B – Most significant Bit of input number B, MSB_SUM – Most significant bit of the 5-bit sum produced by a standard adder circuit.

Answer:

We begin by noting that overflow can only occur when we add two numbers of the same sign. Then we note that the signature of an overflow situation is when the sign of the sum differs from the sign of the two inputs. So we are looking for situations where MSB_A and MSB_B are the same as each other but different from MSB_SUM. This can be accomplished as follows.

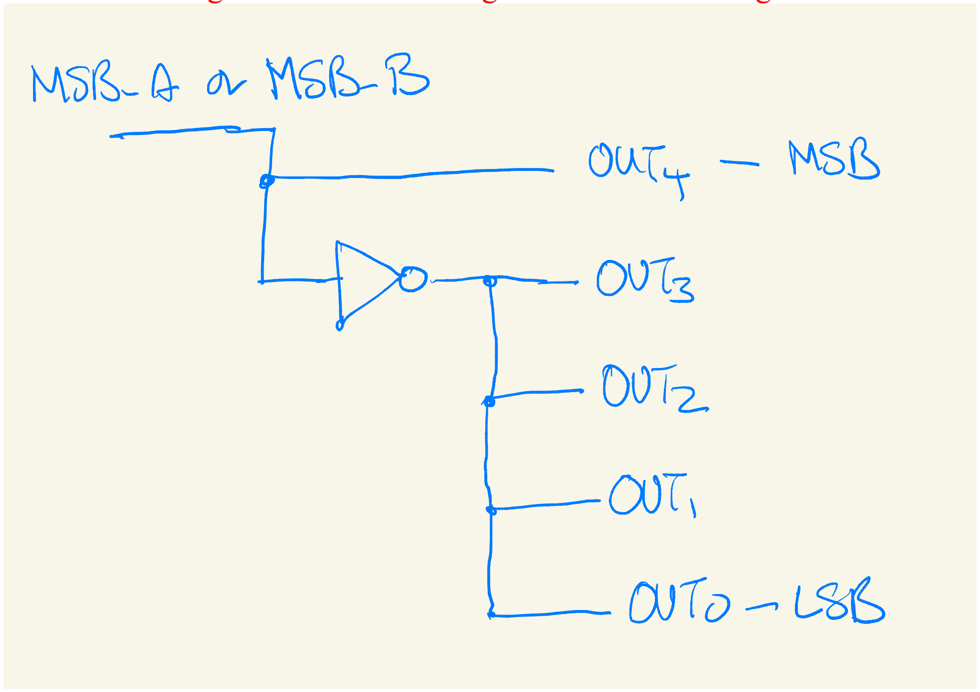


Note that there are many possible answers.

Part 2 {10 pts}: Produce a gate level circuit that produces a 5-bit output corresponding to the 2C result that the saturating adder should produce if an overflow condition is detected when the input numbers, A and B, are added. Your circuit can use any of the following signals as inputs: MSB_A – Most significant bit of input number A, MSB_B – Most significant bit of input number B, MSB_SUM – Most significant bit of the 5-bit sum produced by a standard adder circuit.

Answer:

Here we note that if inputs A and B are both positive the saturated result, if needed, should be the maximum 5-bit 2C value which is 01111. Similarly if A and B are both negative the saturated result, if needed should be the most negative 5-bit 2C value which is 10000. Note that in both cases the MSB of the result is the same as the MSB of the inputs, A or B and the remaining bits are inverses of that. This means that we can implement our solution with a single inverter. Even better, if you notice that in the case of an overflow, MSB_SUM is the inverse of MSB_A/MSB_B you can use that signal instead of adding the inverter resulting in a circuit with no gates at all!

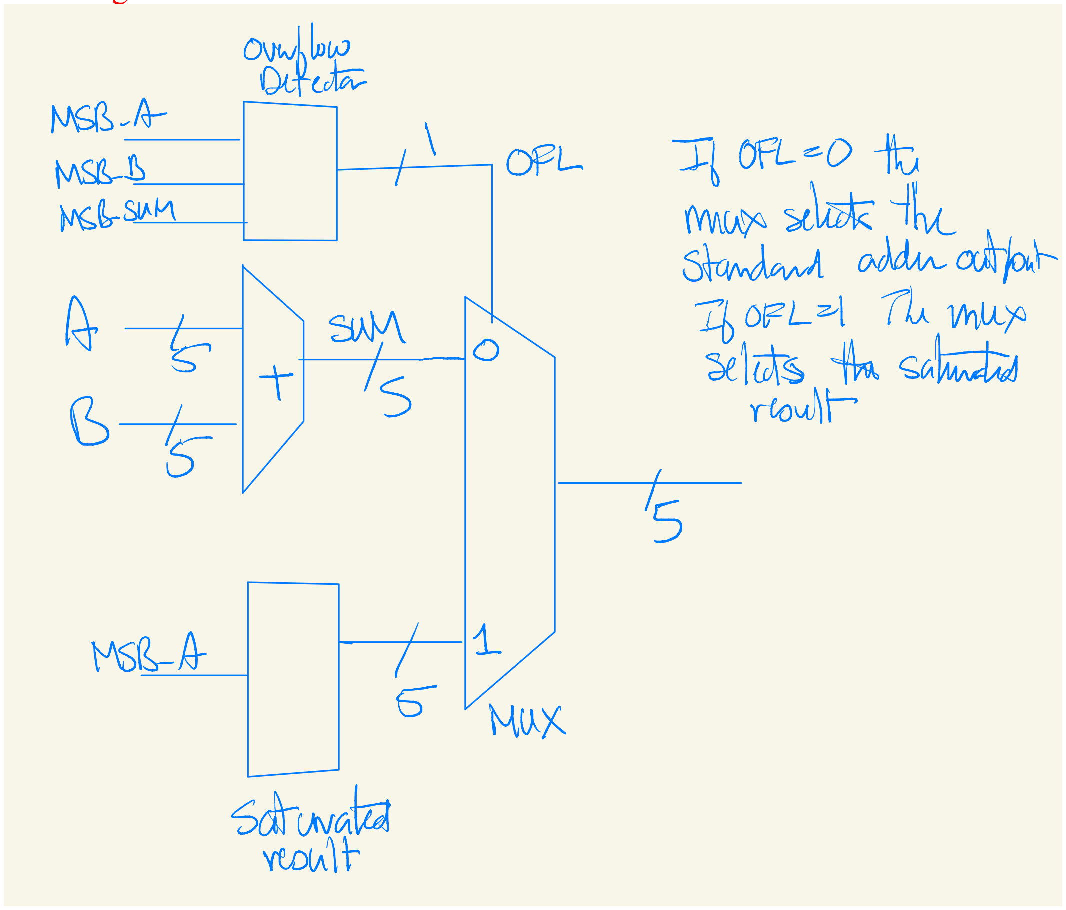


Note that there are many possible answers.

Part 3 {10 pts}: Using your answers to part 1 and part 2 as modules, produce a schematic showing how you would implement the desired saturating adder that takes the 5-bit inputs, A and B, and produces the 5-bit result of the saturating addition. Note you can also use standard 5-bit adders and 5-bit multiplexers as modules in your design. Clearly label each module in your circuit along with any inputs, outputs, or other relevant internal signals.

Answer:

For the saturating adder we proceed by adding the two 5 bit numbers A and B with a standard adder to produce a sum. If there is no overflow this is the result we want to produce. If overflow is detected, we want to output the saturated value. This leads to the following solution.


Overflow Detector
MSB_A
MSB_B
MSB_SUM
1
OFL
A
5
B
5
+
SUM
5
0
1
5
MSB_A
5
MUX
Saturated result
If OFL=0 the mux selects the standard adder output
If OFL=1 The mux selects the saturated result

Note that there are many possible answers.

Question 3 {10 pts}

Having taken CIS 240 you know that common arithmetic operations like addition and multiplication can lead to arithmetic overflow. Can division cause overflow? other than the case of division by zero where the result isn't defined, can the DIV instruction in LC4 result in an arithmetic overflow of any kind? Explain your answer briefly below, just saying yes or no won't get you many points.

Answer:

We accepted two possible correct answers for the question:

1. The DIV instruction performs integer division on 2 values A and B. The point is A/B must produce a result whose magnitude is less than (or equal to) A. A is a valid 2C number so this quotient must also be a valid 2C number because its magnitude is smaller. The conclusion is that the DIV operation cannot produce any kind of overflow as long as you aren't dividing by zero.

2. If you believe that the DIV instruction correctly implements 2C division, one could consider the case where you divide the most negative 16 bit 2C number by -1 resulting in a positive value that cannot be represented in 16 bit 2C. This would be an overflow case.

Question 4

Consider the assembly program shown below.

```
        .CODE               ; This is a code segment
        .ADDR  0x0000       ; Start filling in instructions at address 0x00

        CONST R2, #0        ; Initialize C to 0

LOOP    CMPI R1, #0         ; Compare B to 0
        BRnz END            ; if (B <= 0) Branch to the end

        ADD R2, R2, R0      ; C = C + A
        ADD R1, R1, #-1     ; B =  B - 1

        BRnzp LOOP  ; Go back to the beginning of the loop

END     NOP
```

Part 1 {10 pts}

Use the table below to show what the contents of memory would be once this program is assembled and loaded into PennSim. Each entry should be a 16- bit binary value. We have provided the first entry as a template.

Answer:

Memory Address	Contents (in binary)
0x0000	1001010000000000 CONST R2, #0
0x0001	0010001100000000 CMPI R1, #0
0x0002	0000110000000011 BRnz 3
0x0003	0001010010000000 ADD R2, R2, R0
0x0004	0001001001111111 ADD R1, R1, #-1
0x0005	0000111111111011 BRnzp -5
0x0006	000000xxxxxxxxxx NOP

Part 2 {20 pts}

Each column below shows the state of all registers at the start of each instruction cycle. Simulate the action of the assembly program by filling in the rest of the entries in this table Fill in the NZP entries with N for negative, Z for zero and P for positive. You must fill in the PC and NZP for every clock cycle, for the register values R0-R7 you only need to fill in the values of the registers that have changed from the previous cycle, if any. You should enter all values as decimal numbers, not hex.

Instruction Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
PC	0	1	2	3	4	5	1	2	3	4	5	1	2	3	4
NZP	P	Z	P	P	N	P	P	P	P	N	P	P	P	P	N
R0	-3														
R1	4					3					2				
R2	17	0			-3					-6					-9
R3	0														
R4	0														
R5	0														
R6	0														
R7	0														

Question 5

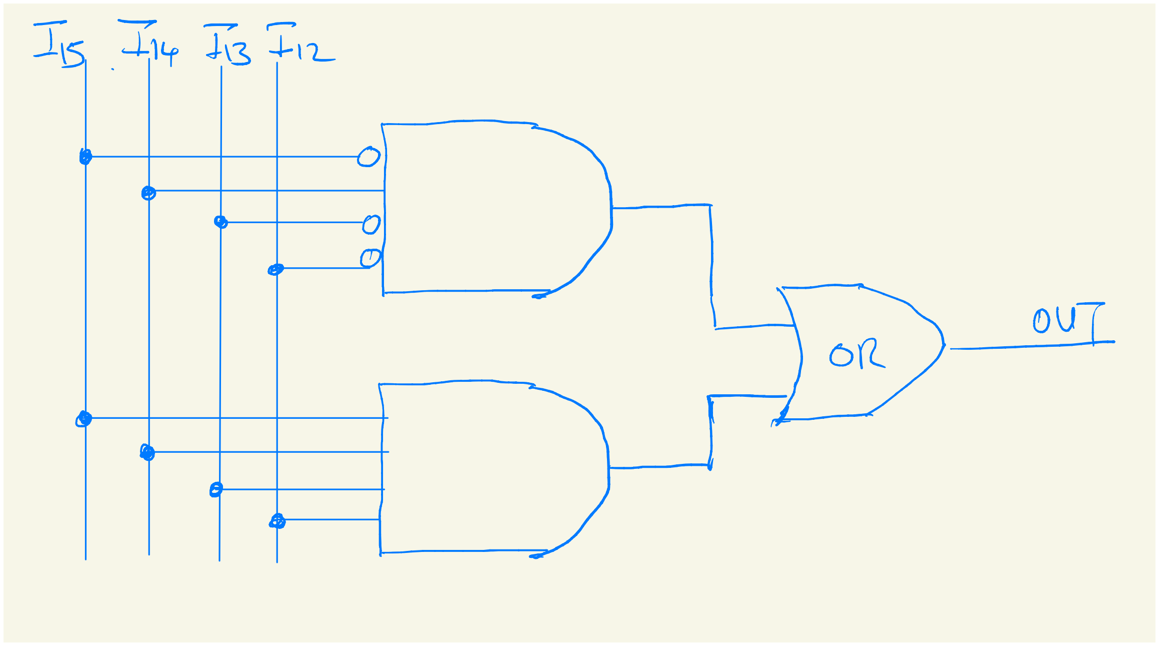
Your job in this question is to design part of the Decoder circuit for our Single Cycle LC4 implementation.

a) **{10 pts}** Produce a PLA circuit that takes as input the relevant bits from the current instruction and produces as output the rdMux.CTL signal.
b) **{10 pts}** Produce a second circuit that generates the rtMux.CTL signal.

Please use the convention $I_{15}, I_{14}, \ldots, I_0$ to refer to bits in the instruction word where I_{15} is the MSB and I_0 the LSB.

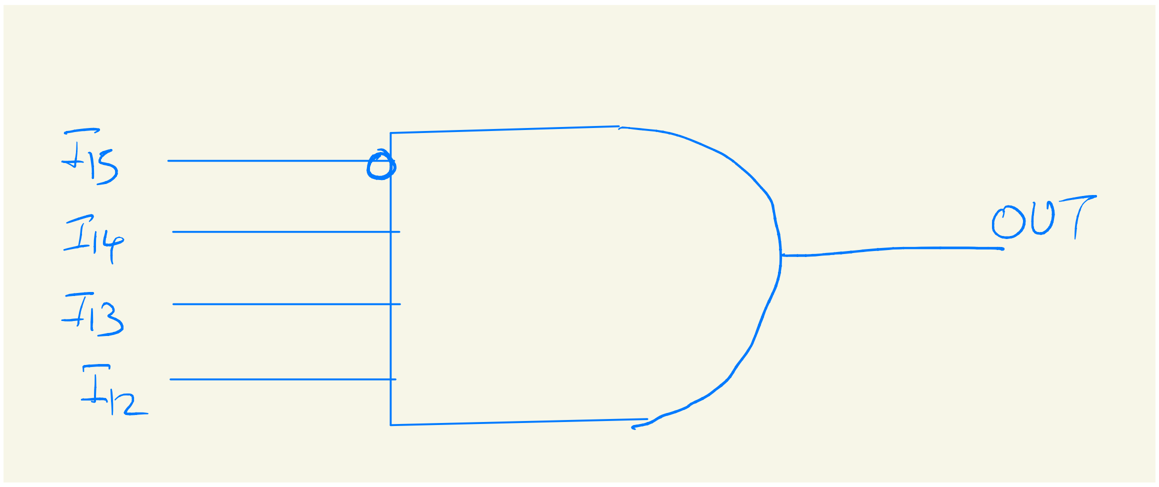
Answer {part a}:

rdMux.CTL must be set to 1 for instructions that implicitly store to R7 namely: JSRR, JSR, and TRAP, Opcodes 0100 and 1111, otherwise it should be 0.



Answer {part b}:

rtMux.CTL only needs to be 1 for 1 instruction, STR, opcode 0111. Otherwise it can be 0.



I_{15}
I_{14}
I_{13}
I_{12}
OUT