University of California, Berkeley – College of Engineering
Department of Electrical Engineering and Computer Sciences
Summer 2018 Instructors: Steven Ho, Nick Riasanovsky 2018-07-25

CS61C MIDTERM 2

***Last** Name (Please print clearly)*	Perfect	
***First** Name (Please print clearly)*	Petra	
Student ID Number	61C	
Circle the name of your Lab TA	**Damon Jonathan Sean Sruthi Emaan Suvansh Sukrit**	
Name of the person to your: Left | Right	Steven	Nick
All my work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who haven't taken it yet. ***(please sign)***		

Instructions

- This booklet contains 10 pages including this cover page. **The back of the exam contains scratch paper and a handout that can be used for scratch work, but will not be graded** (i.e. not even scanned into Gradescope).
- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats, headphones, and watches. Place *everything* except your writing utensil(s), cheat sheet, and beverage underneath your seat.
- You have 80 minutes to complete this exam. The exam is closed book: no computers, tablets, cell phones, wearable devices, or calculators. You are allowed one page (US Letter, double-sided) of *handwritten* notes.
- There may be partial credit for incomplete answers; write as much of the solution as you can.
- Please write your answers within the boxes and blanks provided within each problem!
- For multiple choice questions, please **bubble** in the circle. Failure to do so may impact our ability to grade your exam.

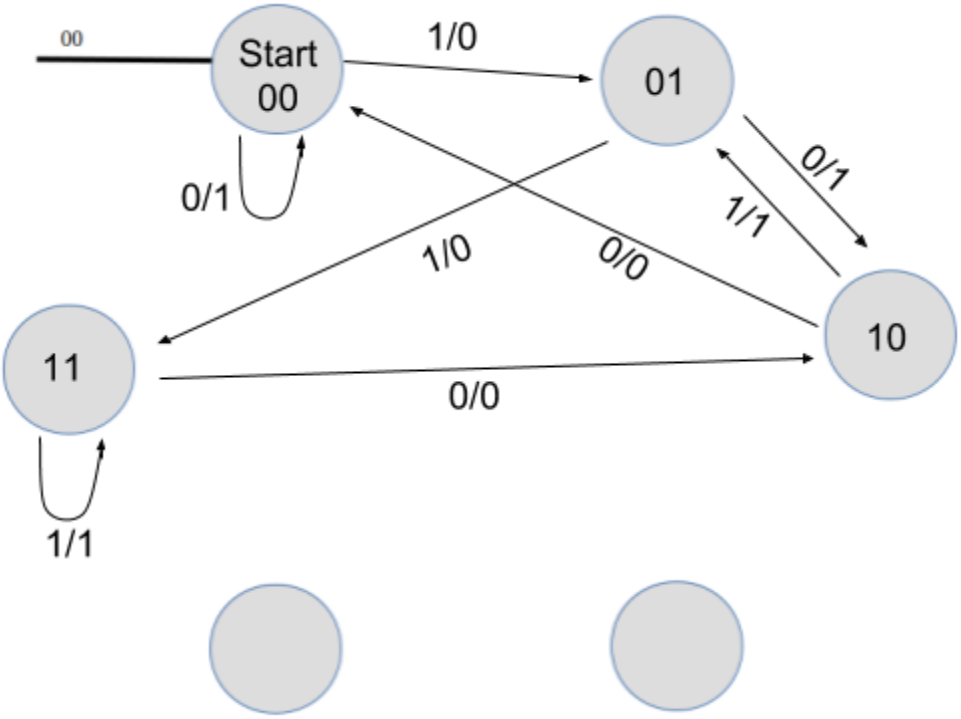
Question	**1**	**2**	**3**	**4**	**5**	**Total**
Possible Points	11	12	20	22	15	80

If you have the time, **feel free to doodle on this front page!**

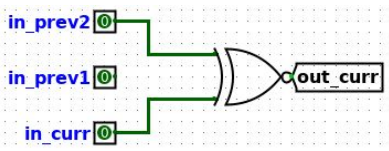
SID: _____

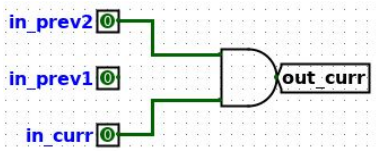
Question 1: **FSMSF** (11 pts)

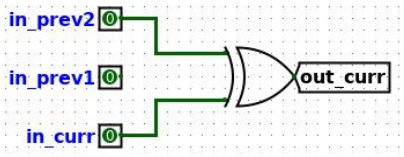
1) Construct an FSM which outputs a 1 when the most recent 3 bits in the input stream are a palindrome. A palindrome is a sequence which is the same when displayed in reverse order. An example of a palindrome is 010, whereas 001 is not. This FSM should continually be updating the output when the input stream changes. For example, when the bitstream is 0010010101, the output is **01001111. The first two outputs are undefined and you should ignore them. Assume the bitstream is always initialized to 00 for you. Your FSM should use the fewest possible number of states. As a reminder you should represent transitions with an arrow, where the tail of the arrow is the origin state and the tip is the destination state. You should also label each transition arrow as `<input>` / `<output>`. (e.g: 0 / 1)

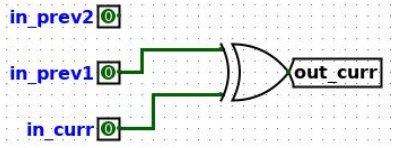


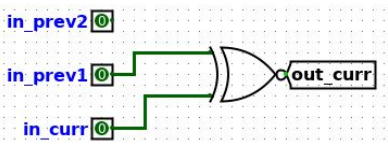
Select the circuit diagram which produces the correct output for the FSM. in_curr is the current input to the FSM (in[i]) , in_prev1 is the previous input (in[i - 1]), and in_prev2 is the input before the previous input (in[i - 2]).

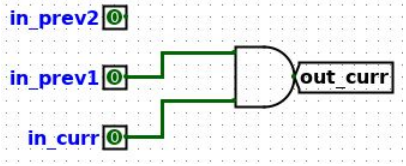
A.


B.


C.


D.


E.


F.


2) Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ

SID: _____

Question 2: Simple Democratic Selection (12 pts)

As the semester is reaching a close, Steven, Nick, and Damon are busy determining the difficulty of the final exam. All three will vote on whether the final should be easy or hard, but the final decision will always be made based on the following rules:

Rule 1. If the vote is unanimously hard or unanimously easy, then it will be hard or easy, respectively.
Rule 2. If Damon disagrees with Steven and Nick, then Damon's vote will be chosen.
Rule 3. If Steven and Nick differ, then the minority vote will be chosen.
Else In all other situations, the outcome can be either easy or hard (i.e. they can be anything)

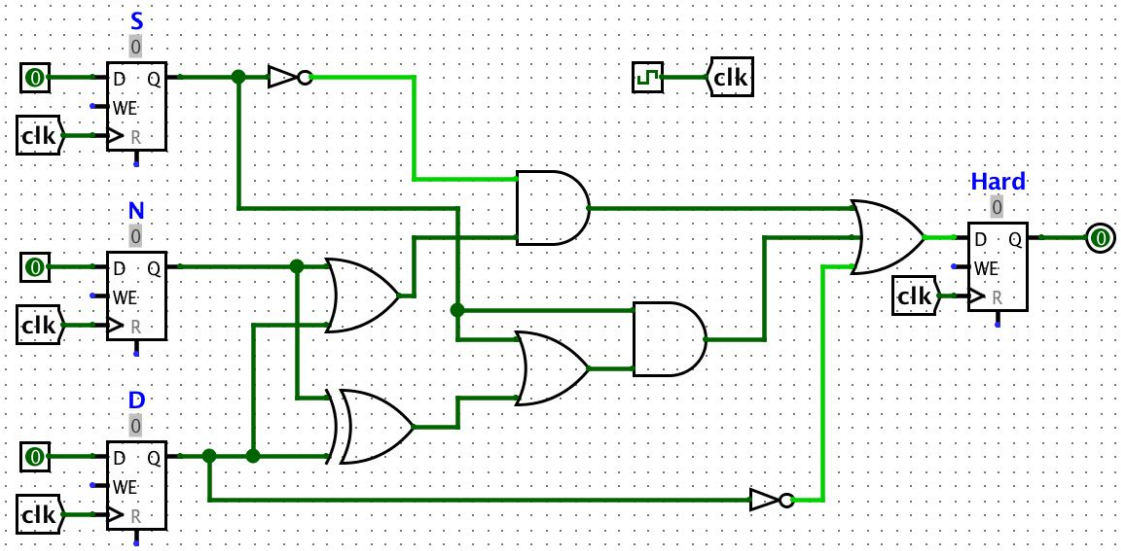
We will represent Steven's vote with the variable **S** which takes on values of 0 (easy) and 1 (hard). Similarly, Nick's vote is represented as **N** and Damon's vote is represented as **D**. For each rule, write out the **simplest** boolean logic expression using these three binary inputs that outputs whether or not the final exam will be easy or hard. Note: the symbol for XOR is ⊕.

Rule 1: SND **Rule2:** $(D \oplus S)\,(D \oplus N)\,D$ *or* $\overline{S \oplus N}\,D$ **Rule3:** $(S \oplus N)\,\overline{D}$

Below is a boolean algebra expression that models this problem. Simplify it into as few gates as possible:

$\overline{S}\,(N+D) \;+\; S(S \;+\; (N \oplus D)) \;+\; \overline{D}$ Simplified: $S(S \;+\; (N \oplus D) \;=\; S\,,\; \overline{S}\,(N+D) \;+\; S \;=\; S \;+\; N \;+\; D,\; 1$

For the next 2 questions, refer to the circuit diagram below which models the unsimplified boolean expression:



You are given that all logic gates have a propagation delay of 5ps, the register clk-to-q delay is 3ps, the register hold time is 11ps, and the setup time is 8ps. What is the critical path delay of this circuit?

Critical path delay: 3 + 5 + 5 + 5 + 5 + 8 = 31 ps

Using the assumptions above, what is the smallest value that the clk-to-q can be and **not** cause a hold-time violation?

clk-to-q: 11 - 5 + 5 = 1 ps

SID: _____

Question 3: Maddi-o and Lui-p (20 pts)

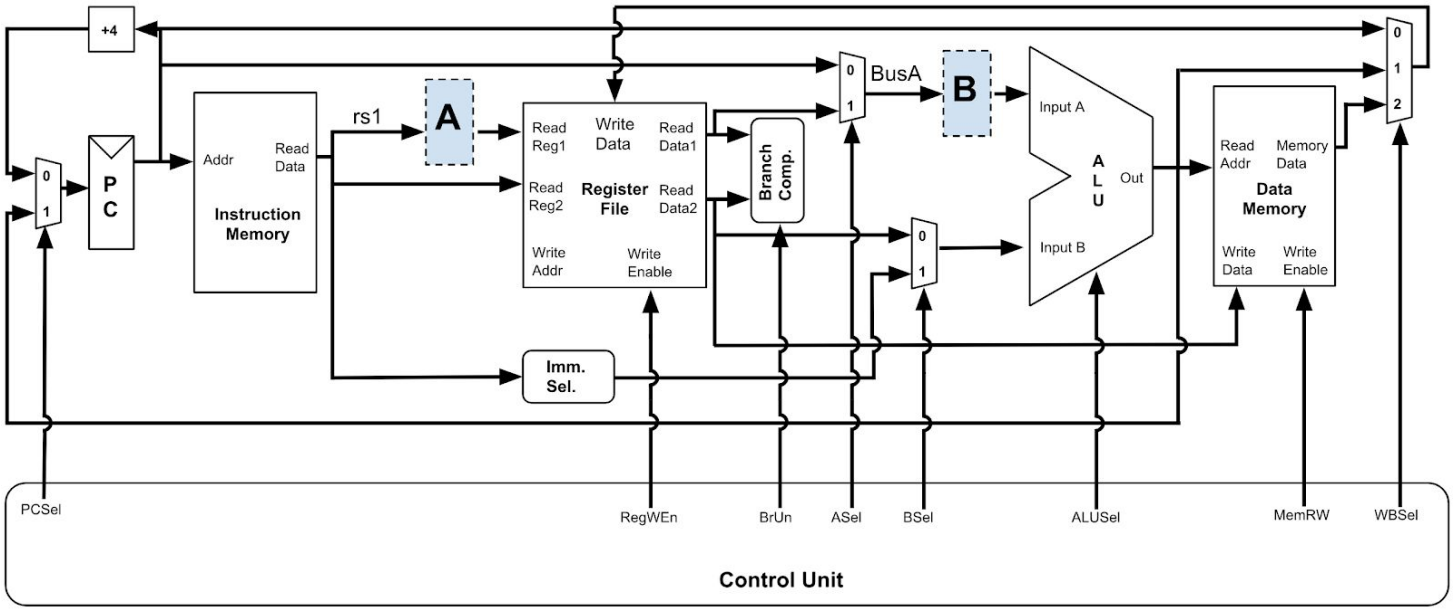
As discussed in lecture, the standard RISC-V approach for loading in a 32-bit immediate into a register is by using the following **`lui-addi`** pair of instructions:

```
lui   rd HIGH_IMM
addi  rd rd LOW_IMM
```

We will explore an alternative method of storing a 32-bit immediate into a register by implementing a new **U-type** instruction named **`luip`,** which will load a 20-bit immediate into the upper 20 bits of the `rd` register and **preserve** the lower 12 bits of the `rd` register. This way, we can use an `addi-luip` pair to load a 32-bit immediate into a register, as shown on the left. On the right is a description of what **`luip`** does:

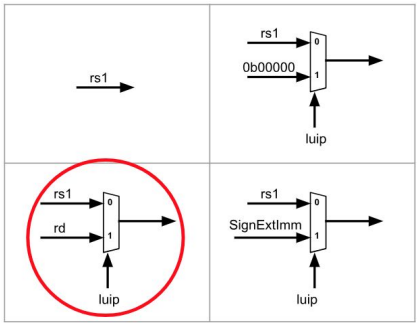
```
addi  rd x0 LOW_IMM                    R[rd] = R[rd] [11:0] + {imm, 12b'0}
luip  rd HIGH_IMM
```

Below is the existing RISC-V datapath as presented in lecture, except there are two boxes labeled “A” and “B” which are areas in the datapath that you will fill in with the necessary hardware to implement **`luip`**.

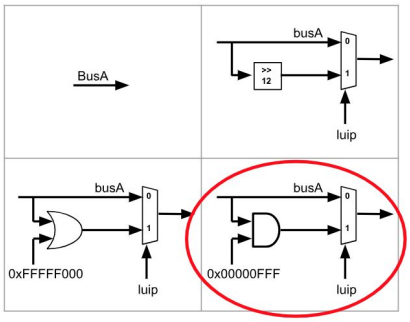


For Parts A and B, please **clearly circle** the correct hardware that would fill their respective dashed boxes above. For Part C, **circle** all the ALUSel operations that could be used by the **`luip`** instruction in the ALU unit.

A.



B.



C.

AND	OR
ADD	SUB
XOR	BSEL

SID: _____

D. Fill out the following control signals that must be generated by the control unit in order to correctly implement **`luip`**. Fill in the bubble for "X" if the exact value of the control signal does not matter (e.g. a value of ⓪ or ① is incorrect when the signal could instead be Ⓧ).

⓪ Signal = 0 ① Signal = 1 ② Signal = 2
Ⓡ Write Disabled Ⓦ Write Enabled Ⓧ Don't Care

`luip`	PCSel	RegWEn	BrUn	ASel	BSel	MemRW	WBSel
1	⓪ ① Ⓧ	Ⓡ Ⓦ Ⓧ	⓪ ① Ⓧ	⓪ ① Ⓧ	⓪ ① Ⓧ	Ⓡ Ⓦ Ⓧ	⓪ ① ② Ⓧ

E. Now that we've added **`luip`** to our instruction set architecture, we need to give it an opcode. We now have 3 U-type instructions: **`auipc`**, **`lui`**, and **`luip`**. The opcode for `auipc` is 0b001 0111 and the opcode for `lui` is 0b011 0111. For this question, we refer to the most significant (leftmost) opcode bit as bit 6 and the least significant (rightmost) opcode bit as bit 0.

You are given the following truth table whose two inputs are two of the opcode bits, and the output is which U-type instruction is being identified by the two opcode bits. Fill out all possible opcode bits that could be the input to the truth table in order to generate the specified outputs (**an output of X means there is no defined instruction with that opcode in our ISA**).

(Hint: the other five bits of the opcode are compared to a **constant** value which identifies the opcode as being a U-type opcode—your task is figuring out which bits can identify which U-type instruction it is.)

Opcode bit(s): ⑥ ⑤ ④ ③ ② ① ⓪	**Opcode bit(s):** ⑥ ⑤ ④ ③ ② ① ⓪	Instruction
0	0	X
0	1	`luip`
1	0	`auipc`
1	1	`lui`

SID: _____

Question 4: RISC IS HAZARDOUS (22 pts)

Consider a RISC-V pipeline in 3 stages with the following specification:

Stage 1	Stage 2	Stage 3
Instruction Fetch/Instruction Decode (IFD)	Execute (EX)	Memory/Write Back (MWB)

It also has the following details:

- No Forwarding
- No reading and writing to the same register in the same cycle
- Reading and writing to the same register is considered a data hazard

Consider the following piece of code

```
1.        add t1 x0 x0
2.        add t2 x0 x0
3.        addi a0 x0 2
4.        slli a0 a0 2
5.  L2:   bge t1 a0 End
6.        add t3 sp t1
7.        lw t3 0(t3)
8.        add t2 t2 t3
9.        addi t1 t1 4
10.       j L2
    .
    .
    .
End:
```

1) Assume that instead of branch prediction, the CPU always stalls to resolve a control hazard. How many stall(s) are necessary for each control hazard **each time** it is encountered? You may not need all boxes.

Line Number	# Stalls/Encounter
5	1
10	1

2)

a) Where are the data hazards that produce stall(s)? You should describe each hazard as a tuple, (A, B), where instruction A is the the instruction that triggered a data hazard in instruction B. Place A's line number in the left box and B's line number in the right box You may not need all boxes.

Line of Instruction that first accesses the data	Line of instruction that must be stalled
3	4
4	5
6	7
7	8
9	5

b) How many **total cycles** will it take to complete the code above? Assume the pipeline is cleared upon reaching End. In addition, assume that there is perfectly accurate branch and jump prediction in the IFD stage. Thus, **ONLY CONSIDER STALLS DUE TO DATA HAZARDS. ASSUME THERE ARE NO STALLS FOR STRUCTURAL OR CONTROL HAZARDS.** Remember to **calculate the total number of cycles for fully executing the code**. A workspace was provided with this exam for you to show your work, which will only be graded if your final answer is not correct.

Total Cycles: 33 (Remember we said there are no stalls for control hazards)

SID: _____

Now assume that our pipeline has full forwarding along with perfect branch/jump prediction done in the IFD stage.

3) What are the data hazards that remain? How many cycles must be stalled **each time** this hazard is encountered? If there are two instructions with the same data hazard and both are resolved by the first stall then only list the first instruction as needing to be stalled. The left box(es) should be a subset of the value(s) you had in the rightmost box(es) in 2a. You may not need all boxes.

Line of instruction that must be stalled	# Stalls/Encounter
8	1

SID: _____

Question 5: C.R.E.A.M. (15 pts)

---

1) A machine has an 8 way set associative cache with 512 B of data. The size of each block is 16 B and there are 8 MiB of main memory. How large are the tag, index, and offset fields for an address on this machine using this cache?

Tag: 17 Index: 2 Offset: 4

2) Now we have a different machine with two caches, an L1 and an L2 cache. Both caches are direct mapped caches. The L1 cache can hold 256 B of data and the L2 cache can hold 4 KiB of data. Assume the following code is run on this machine:

```
#define ARR_SIZE 2048

uint16_t sum (uint16_t *arr) {
    total = 0;
    for (int i = 0; i < ARR_SIZE; i++) {
       total += arr[i];
    }
    return total;
}
```

This produces a hit rate (HR) of 7/8 for the L1 cache and 3/4 for the L2 cache. Given that `arr` is a block aligned address and `sizeof (uint16_t) == 2`:

A. What is the blocksize of the L1 cache **in bytes** that produces its hit rate?

L1_BLOCKSIZE: 16 B

SID: _____

B. Use the variable **Y** to represent the answer to part A. What is the blocksize of the L2 cache **in bytes** that produces its hit rate? Express your answer as a function of **Y** and NOT as a single number.

L2_BLOCKSIZE: 4 * Y (64 B)

Recall that the L1 HR is 7/8 and the L2 HR is 3/4. If the L1 Cache has a hit time of 2 cycles, the L2 cache has a hit time of 8 cycles, and main memory has a hit time of 96 cycles:

3) How many total cycles are spent accessing memory on this piece of code? Express your answer in the form $\mathbf{C} * \mathbf{2}^{\mathbf{i}}$, where **C** is an integer not divisible by 2.

TOTAL_CYCLES: 2048 * (2 + 1/8 * (8 + 1/4 * 96)) = 2^11 * (2 + 8/8 + 96/32) = 2^11 * (6) = 3 * 2^12

4) If we change the L1 cache from being direct mapped to being fully associative with LRU, how does its HR change on the same code?

Ⓐ Increases Ⓑ Decreases Ⓒ No Change Ⓓ Impossible to tell