ECE 198JL Third Midterm Exam
Fall 2013

Tuesday, November 12th, 2013

Name: ____ NetID: ________________

Discussion Section:

10:00 AM	[ ] JD1	
11:00 AM	[ ] JD2	
12:00 PM	[ ] JD7	
1:00 PM	[ ] JD9	[ ] JDA
2:00 PM	[ ] JDB	
3:00 PM	[ ] JDC	
4:00 PM	[ ] JD8	

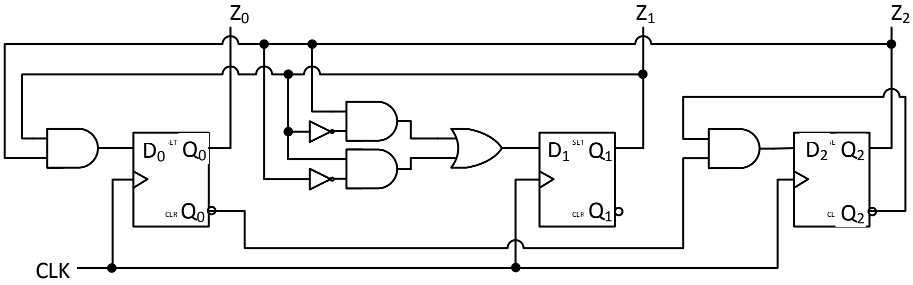
- **Be sure your exam booklet has 11 pages.**
- **Be sure to write your name and lab section on the first page.**
- **Do not tear the exam booklet apart; you can only detach the last page.**
- **We have provided LC-3 instructions set at the back.**
- **Use backs of pages for scratch work if needed.**
- **This is a closed book exam. You may not use a calculator.**
- **You are allowed one handwritten 8.5 x 11" sheet of notes.**
- **Absolutely no interaction between students is allowed.**
- **Be sure to clearly indicate any assumptions that you make.**
- **The questions are not weighted equally. Budget your time accordingly.**
- **Don’t panic, and good luck!**

Problem 1	10 points:	____________
Problem 2	7 points:	____________
Problem 3	14 points:	____________
Problem 4	11 points:	____________
Problem 5	12 points:	____________
Problem 6	22 points:	____________
Problem 7	8 points:	____________
Problem 8	16 points:	____________

Total	100 points:	____________

Problem 1 (10 pts): Sequential circuit analysis

Consider the sequential circuit shown below that has three internal state bits, $Q_2Q_1Q_0$, and three external outputs, $Z_2Z_1Z_0$, that match the internal state bits, that is, $Z_i = Q_i$.



1. Write the flip-flop excitation equations for the D_2, D_1, and D_0 flip-flops as functions of the current state $Q_2Q_1Q_0$:

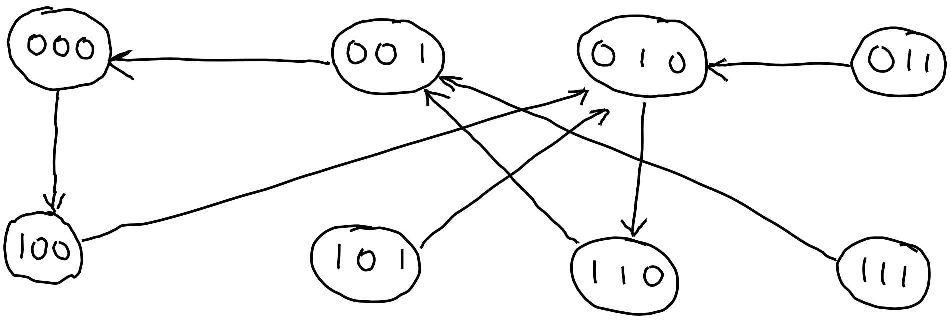
$D_2 = \overline{Q_2}\,\overline{Q_0}$ $D_1 = Q_2\overline{Q_1} + \overline{Q_2}Q_1$ $D_0 = Q_2\,Q_1$

(Observe: $D_1 = Q_2 \oplus Q_1$)

2. Complete the next-state table

Current state			Next state		
Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	0	0	1
1	1	1	0	0	1

3. Draw the state transition diagram for this FSM.

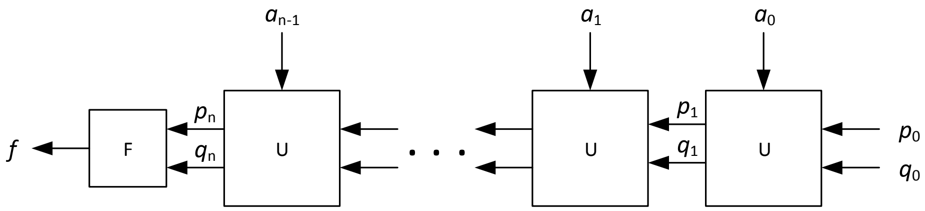


4. Explain the function of this FSM in one sentence.

Counter that goes through a sequence of 5 states

Problem 2 (7 pts): Serial design

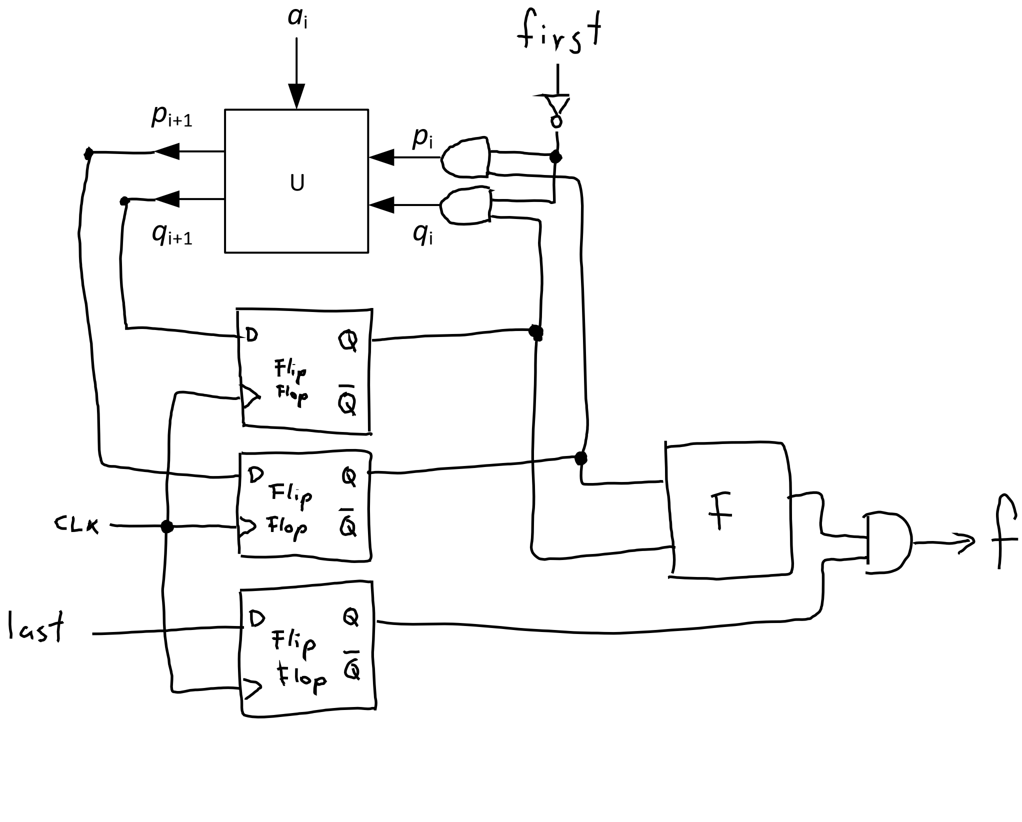
On Midterm 2 you designed a bit-slice circuit that checks whether an unsigned integer $A=a_{n-1}a_{n-2}..a_1a_0$ is a power of 2, starting with the least significant bit:



1. How many flip-flops are needed to construct a *serial* power-of-two checker circuit using a Moore machine model and using circuit U as is?

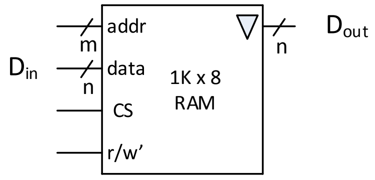
Answer: Two

2. Re-draw the bit-sliced design shown above as a serial design using a Moore machine model. Besides adding storage elements, you also need to add a circuit that resets p_iq_i inputs to 00 when i=0 and a circuit that outputs f when i=n-1, or 0 otherwise. You can assume that two additional input signals are supplied: *first*=1 iff i=0 and *last*=1 iff i=n-1. You can also use circuit F from the above implementation as a black box.



Problem 3 (14 pts): RAM

Shown to the right is a 1K x 8 RAM. (1K = 2^{10}.)

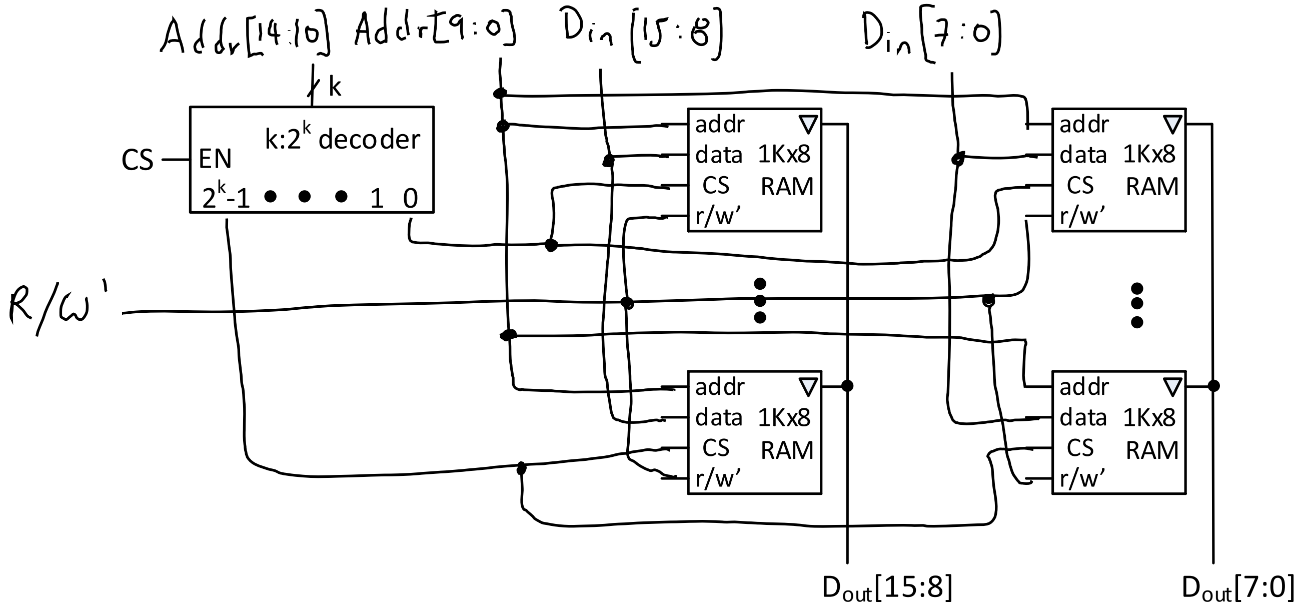


1. How many bits do the **addr**, **data**, **CS**, and **r/w'** ports require?

Answer: addr: 10 CS: 1

data: 8 r/w': 1

2. Using 1K x 8 RAM chips, implement a 32K x 16 RAM. Each 1K x 8 RAM chip has inputs **data**, **addr**, **CS**, and **r/w'** and an output gated by a tri-state buffer. Finish the implementation by drawing the missing connections and labeling all newly added wires. (You do not need to draw all the rows, they are shown as " ... ", but be sure the pattern is clear.) The RAM output wires and CS are already drawn for you.



3. How many rows of 1K x 8 RAM chips are needed and what is the value of *k*?

Number of rows: $2^5 = 32$ *k* = 5

4. Suppose you wish to store the decimal value -1 in memory at the address 1024. In which row (indexing from 0) of 1K x 8 RAM chip(s) will this value be stored?

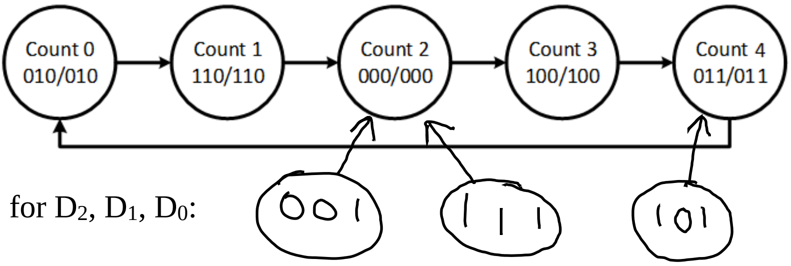
Answer: Row 1 $1024_{10} = \underbrace{000\ 01}_{Addr[14:10]}\underbrace{00\ 0000\ 0000_2}_{Addr[9:0]}$

5. What input values must be provided to your 32K x 16 RAM in order to properly store this value at the address 1024? Write the **addr** and **data** values for your 32K x 16 RAM in **hexadecimal**.

addr: x0400 data: xFF FF

Problem 4 (11 pts): Counter design

Using conventional sequential circuit design techniques, implement a 3-bit synchronous counter with negative-edge triggered D flip-flops that counts in the following $Q_2Q_1Q_0$ sequence:



1. Show K-maps for D_2, D_1, D_0:

D_2

Q_2 \ Q_1Q_0	00	01	11	10
0	1	X	0	1
1	0	X	X	0

D_1

Q_2 \ Q_1Q_0	00	01	11	10
0	0	X	1	1
1	1	X	X	0

D_0

Q_2 \ Q_1Q_0	00	01	11	10
0	0	X	0	0
1	1	X	X	0

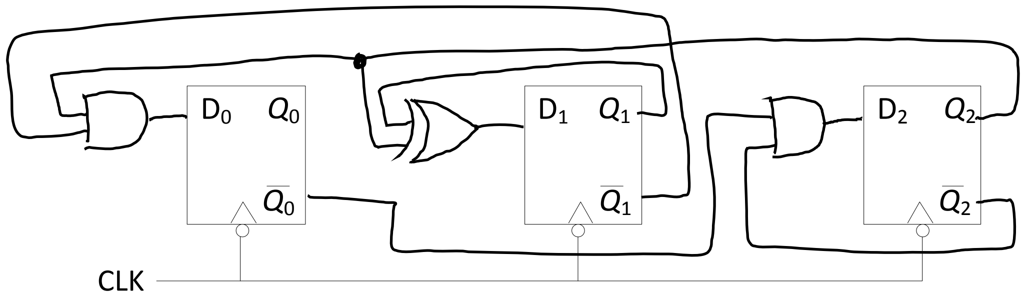
2. Write Boolean expressions for D_2, D_1, D_0 in min SOP form:

$D_2 = \overline{Q_2}\,\overline{Q_0}$

$D_1 = Q_2\overline{Q_1} + \overline{Q_2}Q_1$ $(= Q_2 \oplus Q_1)$

$D_0 = Q_2\overline{Q_1}$

3. Draw the circuit using as few additional gates as possible.



4. Is your counter *self-starting*? Explain - and be specific to *your solution*. (Self-starting means that no matter in what state the counter starts, it always converges to produce the correct counting sequence.)

Yes, As seen above in the complete state diagram, no matter the initial state, the FSM will converge to the desired sequence

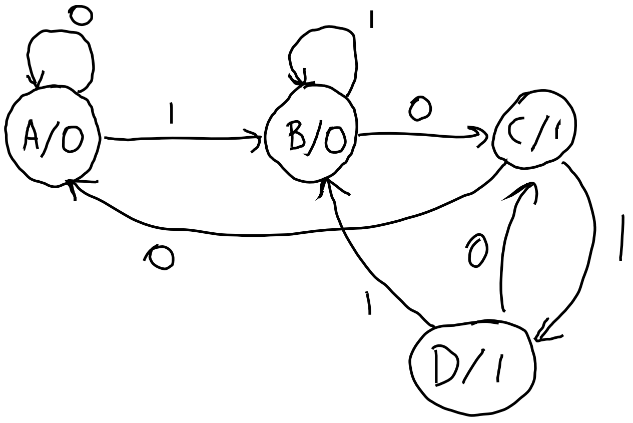
Problem 5 (12 pts): Sequence recognizer design

Design a sequence recognizing FSM with input *x* and output *z* such that the output is 1 if and only if pattern **10** or **101** has been detected in the input stream. Make sure to detect overlapping patterns.

Example:

Input:	0	0	1	0	1	0	1	1	0	1	1	0	1	0	1	1	0	1	0. . .
Output:	0	0	0	1	1	1	1	0	1	1	0	1	1	1	1	0	1	1	1. . .

1. Draw the Moore state diagram, using as few states as possible. (Solutions with more than 5 states are not acceptable.) Let A be the start state. Each edge must be labeled with *x* and each node (state) must be labeled with *name*/*z*.



2. Write down your *state meanings*, e.g., "A – start state":

A : start state

B: last 2 inputs were "01" or "11"

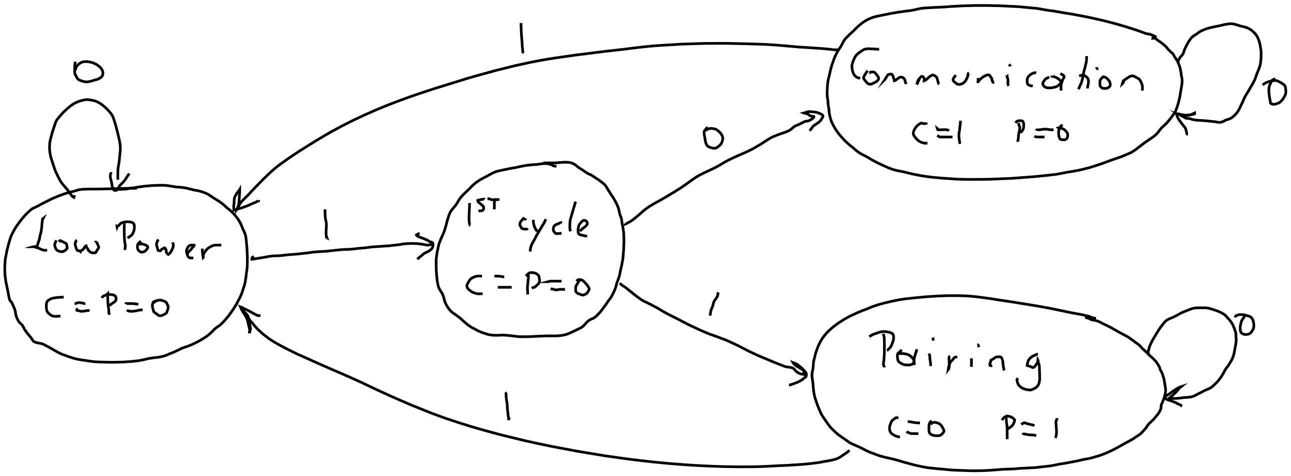
C : sequence "10" seen in last 2 inputs

D : sequence "101" seen in last 3 inputs

Problem 6 (22 pts): Bluetooth headset FSM

A wireless Bluetooth headset for a cell phone has one button (B = 1, when pressed) to control all of its functionality. When the device is “off,” the system is in a low power state waiting for the user to turn it on. If the headset is “off” and the user presses the system button for one cycle, the headset will turn “on” and tell the cell phone that it is ready for communication by setting signal C = 1. If the headset is “off” and the user presses the system button for two cycles, the headset will turn on into “pairing” mode and broadcast a pairing signal by setting P = 1. If the user presses the system button while it is on or in pairing mode, then the headset will turn off. Note: Each part will be graded based on your previous section, so if you cannot perfectly solve part 1, finish the other sections based on whatever solution you find for Part 1.

1. Design a Moore FSM implementing this system. Give your states meaningful names so that we can comprehend your design intent. Sketch the complete state transition diagram for your FSM, specifying input B for each transition and outputs C and P for each state. (*You do not need more than 4 states or a counter. Carefully consider how you will count the cycles that the button is pressed.*)



2. Fill in the truth table below by assigning internal state bit representations S_1S_0 and output values C and P for each of your states from Part 1 and then filling in the values for the next-state variables S_1^+ and S_0^+ for each combination of S_1S_0B. This part will be graded based on your FSM in part 1.

State name and meaning	Current state		External Input B	Next State		External Outputs	
	S_1	S_0	B	S_1^+	S_0^+	C	P
Low Power	0	0	0	0	0	0	0
			1	0	1		
1^{ST} cycle	0	1	0	1	0	0	0
			1	1	1		
Communication	1	0	0	1	0	1	0
			1	0	0		
Pairing	1	1	0	1	1	0	1
			1	0	0		

3. Using the next state table that you wrote in Part 2, fill in the K-maps for S_1^+, S_0^+, C, and P and derive **minimal SOP** Boolean expressions for these variables.

		S_1	
		0	1
S_0	0	0	1
	1	0	0

C = $S_1 \overline{S_0}$

		S_1	
		0	1
S_0	0	0	0
	1	0	1

P = $S_1 S_0$

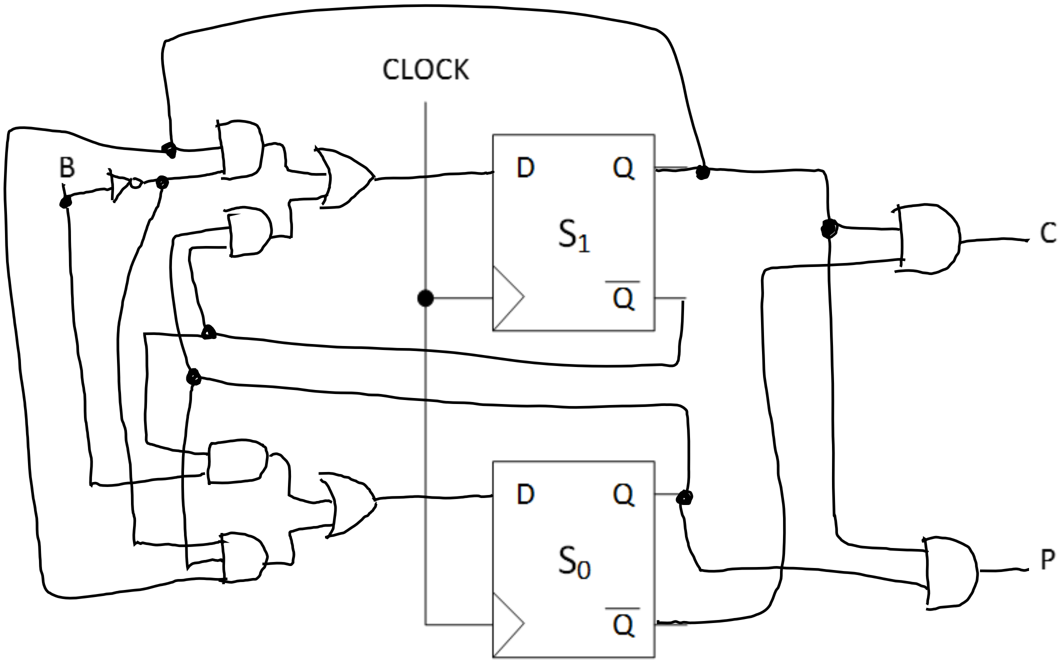
		S_1S_0			
		00	01	11	10
B	0	0	1	1	1
	1	0	1	0	0

S_1^+ = $\overline{S_1} S_0 + \overline{B} S_1$

		S_1S_0			
		00	01	11	10
B	0	0	0	1	0
	1	1	1	0	0

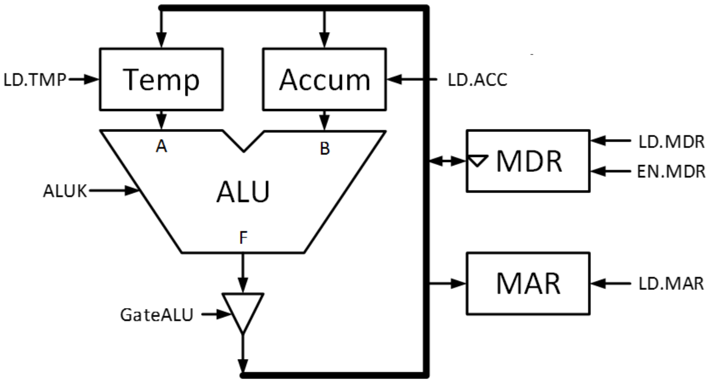
S_0^+ = $B \overline{S_1} + \overline{B} S_1 S_0$

4. Add gates as necessary to complete the implementation of your FSM using the two positive edge-triggered D flip-flops shown below.



Problem 7 (8 pts): Register transfer

Digital Signal Processing units, like those found in your cell phone, are specialized architectures that use an Accumulator register (Accum) to store the results of all arithmetic operations. The architecture also has a Temporary register (Temp) to store a second operand. The ALU, Memory Data Register (MDR), and Memory Address Register (MAR) are used for the same purpose as in the LC-3. The ALU function table is shown below. The ALU and MDR can both send data to the system bus but only as allowed by the GateALU and EN.MDR signals, respectively. When EN.MDR is 1, MDR's content is output on the bus. When LD.MDR is 1, MDR stores value from the bus. When GateALU is 1, ALU's output is connected to the bus.



ALUK	Operation	Explanation
00	Pass A	F = A
01	Pass B	F = B
10	Clear	F = 0
11	Multiply-Accumulate	F = A*B + B

1. The Accumulator is part of the processing unit of a von Neumann architecture.

2. Assign values to the control signals to execute the following RTL instructions. If the RTL instruction cannot be implemented, write "IMPOSSIBLE." For the last row, determine what RTL instruction is implemented by the selected control signals.

RTL Instruction	CONTROL SIGNALS						
	LD.TMP	LD.ACC	LD.MDR	LD.MAR	GateALU	EN.MDR	ALUK
MAR ← Accum	0	0	0	1	1	0	01
Accum ← Temp * Accum + Accum	0	1	0	0	1	0	11
Accum ← MAR							
Temp ← MDR	1	0	0	0	0	1	XX
MDR ← 0	0	0	1	0	1	0	10

Impossible

Problem 8 (16 pts): LC-3 ISA

The following LC-3 program fragment, represented as three hexadecimal numbers, is stored in memory at the indicated locations and the following values are stored in registers:

address	instruction
…	
xC100	xA801
xC101	x1322
xC102	xC100
…	

register	value	
R0	x4005	
R1	~~x4004~~	x A803
R2	x4003	
R3	x4002	
R4	~~x4001~~	X A801

1. Re-write three instructions in binary representation and provide their corresponding RTL. (*Note: formats of the entire LC-3 instruction set are provided at the end of the exam booklet.*)

address	instruction	binary instruction	RTL (be specific to this instruction)
xC100	xA801	1010 1000 0000 0001	R4 ← M[M[xC102]]
xC101	x1322	0001 0011 0010 0010	R1 ← R4+2
xC102	xC100	1100 0001 0000 0000	PC ← R4

2. Assuming PC is initially set to xC100, trace the execution of the given program segment for three instruction cycles, filling in the table below. Write down the values stored in the PC, IR, MAR, MDR, N, Z, and P registers **at the end of each instruction cycle**. Values for PC, IR, MAR, and MDR should be written in hexadecimal. Values for N, Z, and P should be written in binary.

PC	**IR**	**MAR**	**MDR**	**N**	**Z**	**P**
xC101	xA801	xC100	xA801	1	0	0
xC102	x1322	xC101	x1322	1	0	0
xA801	xC100	xC102	xC100	1	0	0

3. What hexadecimal value will be stored in R1 after the three instruction cycles?

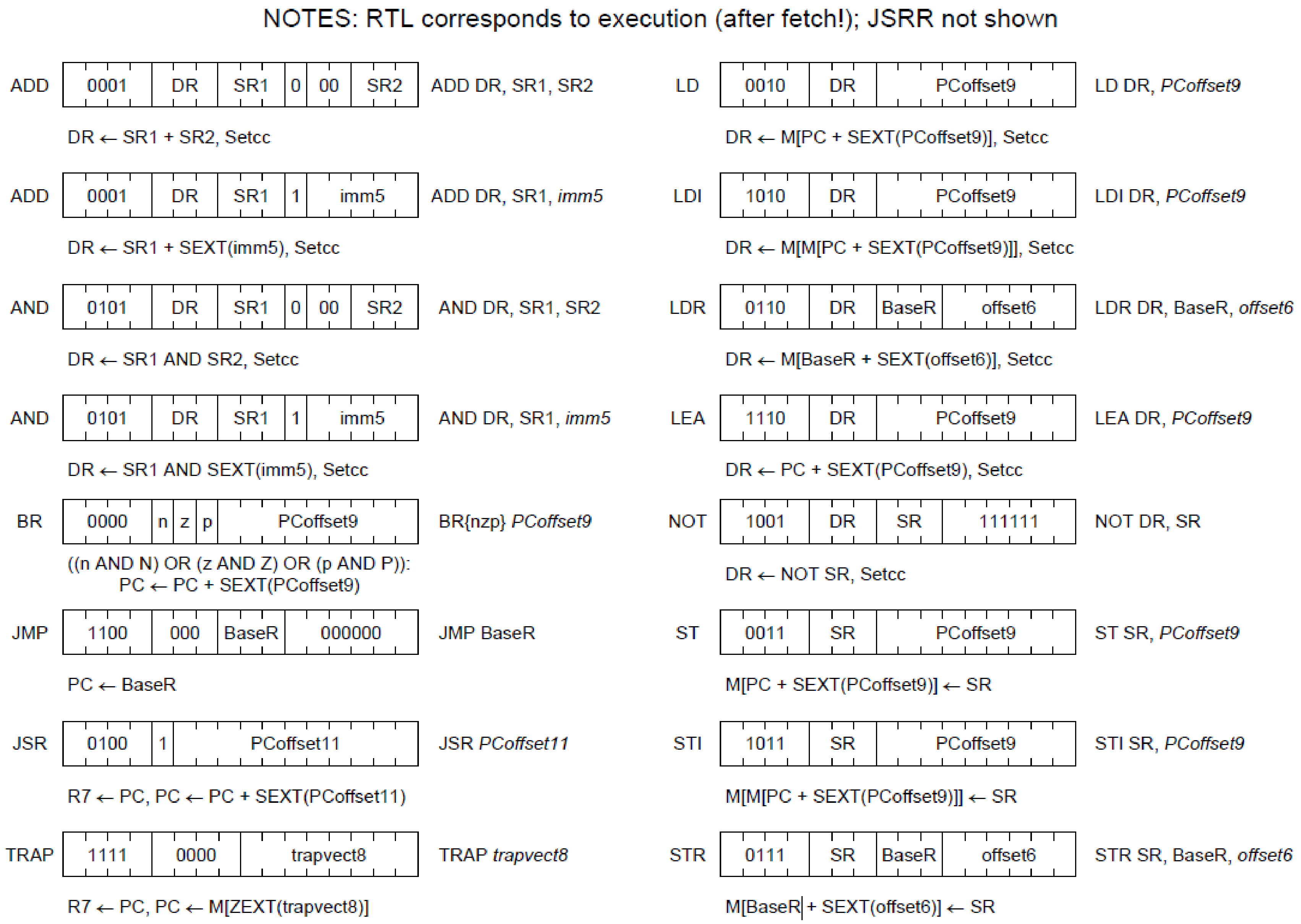
Answer: xA803

4. What is the address of the next instruction to be executed after the three instruction cycles?

Answer: xA801

5. How many memory reads will take place during these three instruction cycles, including the instruction FETCH?

Answer: Five : fetch xC100, read M[xC102], read M[M[xC102]], fetch xC101, fetch xC102



NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

ADD	0001 DR SR1 0 00 SR2	ADD DR, SR1, SR2	DR ← SR1 + SR2, Setcc
ADD	0001 DR SR1 1 imm5	ADD DR, SR1, *imm5*	DR ← SR1 + SEXT(imm5), Setcc
AND	0101 DR SR1 0 00 SR2	AND DR, SR1, SR2	DR ← SR1 AND SR2, Setcc
AND	0101 DR SR1 1 imm5	AND DR, SR1, *imm5*	DR ← SR1 AND SEXT(imm5), Setcc

BR	0000 n z p PCoffset9	BR{nzp} *PCoffset9*	((n AND N) OR (z AND Z) OR (p AND P)):
			PC ← PC + SEXT(PCoffset9)

JMP	1100 000 BaseR 000000	JMP BaseR	PC ← BaseR
JSR	0100 1 PCoffset11	JSR *PCoffset11*	R7 ← PC, PC ← PC + SEXT(PCoffset11)
TRAP	1111 0000 trapvect8	TRAP *trapvect8*	R7 ← PC, PC ← M[ZEXT(trapvect8)]
LD	0010 DR PCoffset9	LD DR, *PCoffset9*	DR ← M[PC + SEXT(PCoffset9)], Setcc
LDI	1010 DR PCoffset9	LDI DR, *PCoffset9*	DR ← M[M[PC + SEXT(PCoffset9)]], Setcc
LDR	0110 DR BaseR offset6	LDR DR, BaseR, *offset6*	DR ← M[BaseR + SEXT(offset6)], Setcc
LEA	1110 DR PCoffset9	LEA DR, *PCoffset9*	DR ← PC + SEXT(PCoffset9), Setcc
NOT	1001 DR SR 111111	NOT DR, SR	DR ← NOT SR, Setcc
ST	0011 SR PCoffset9	ST SR, *PCoffset9*	M[PC + SEXT(PCoffset9)] ← SR
STI	1011 SR PCoffset9	STI SR, *PCoffset9*	M[M[PC + SEXT(PCoffset9)]] ← SR
STR	0111 SR BaseR offset6	STR SR, BaseR, *offset6*	M[BaseR + SEXT(offset6)] ← SR