CS-2110 A/B/C Quiz 3 (C)

Aaryan Vinay Potdar

TOTAL POINTS

75.5 / 100

QUESTION 1

1 1ai **1 / 3**

+ 3 pts Correct (010110011)

+ 2 pts correct but not 9 bits

✓ **+ 1 pts** *incorrect (off by 1)*

+ 2 pts correct but not binary

+ 0 pts incorrect

QUESTION 2

2 1aii **3 / 3**

✓ **+ 3 pts** *Correct (111111001)*

+ 2 pts correct but not in 9 bits

+ 1 pts incorrect (off by 1)

+ 2 pts correct but not binary

+ 0 pts incorrect

QUESTION 3

3 1b **5 / 5**

✓ **+ 5 pts** *Correct*

![Screenshot_2023-10-22_at_4.41.56_PM.png](/files/4de40b02-28d8-4307-8343-a03c73a49833)

+ 4 pts Did not count 0 as a multiple of 32

+ 3 pts computed opposite

+ 3 pts Used cascading NORs instead of NOR5

+ 3 pts off by one input on either side

+ 1 pts correct but using gates other than NOR

+ 0 pts incorrect

+ 0 pts Click here to replace this description.

QUESTION 4

4 1c **0 / 8**

+ 3 pts Correct (both are valid)

+ 5 pts explanation correct

(Ex. Incrementing the PC can be done at any point in the fetch microstates because it does not share wires with the bus or memory unit)

✓ **+ 0 pts** *Incorrect (selects only one option or neither)*

QUESTION 5

5 1d **6 / 6**

✓ **+ 6 pts** *Correct (0001 0100 1000 0010 aka ADD R2, R2, R2)*

+ 5 pts 3/3 registers correct, opcode incorrect

+ 2 pts opcode correct (0001 aka ADD)

+ 2 pts 2/3 registers correct

+ 0 pts Incorrect

QUESTION 6

6 1e **10 / 10**

✓ **+ 10 pts** *Correct*

LEA: 1110 (DR) (all zeros or all ones)

JMP: 1100 000 (Same reg as LEA) (all zeros)

+ 6 pts LEA with any DR and offset == 0 or -1 1110 (DR) (all zeros or all ones)

+ 4 pts JMP with BR == DR of LEA instruction JMP: 1100 000 (Same reg as LEA) (all zeros)

+ 2 pts JMP with BR NOT EQUAL to DR of LEA instruction

- 2 pts instruction encoded with minor mistake (ex. all 1s at the end of JMP instead of 0s) but otherwise correct

+ 0 pts incorrect

QUESTION 7

7 1fi **3 / 3**

✓ **+ 1 pts** *Correct*

✓ **+ 2 pts** *explanation correct*

- They are edge triggered registers

- OR built out of D-flip flops (indicating edge-triggered)

- OR sequential logic

+ 0 pts incorrect

QUESTION 8

8 1fii **3 / 3**

✓ **+ 1 pts** *Correct (False)*

✓ **+ 2 pts** *explanation correct*

- Definition of Von Neumann model (instructions and program data share memory)

+ 0 pts incorrect

QUESTION 9

9 1fiii **0 / 3**

+ 1 pts Correct (False)

+ 2 pts explanation correct

- The LC3 uses bit 5 of the IR to control the SR2mux so you only need 1 microstate for both

✓ **+ 0 pts** *incorrect*

QUESTION 10

10 1fiv **3 / 3**

✓ **+ 1 pts** *Correct (False)*

✓ **+ 2 pts** *explanation correct*

- That would cause multiple values on the bus leading to a short circuit

+ 0 pts incorrect

QUESTION 11

11 1fv **3 / 3**

✓ **+ 1 pts** *Correct (True)*

✓ **+ 2 pts** *explanation correct*

- R1 AND all 1s == R1

+ 0 pts incorrect

QUESTION 12

12 2a **6 / 6**

✓ **+ 6 pts** *Correct*

![Screenshot_2023-10-22_at_4.48.06_PM.png](/files/f37adf37-1fa4-42db-8868-a9acbde17543)

Row 1

+ 0.25 pts Mnemonic (opcode) correct (NOT)

+ 0.5 pts operands correct (R7, R0)

Row 2

+ 0.25 pts Mnemonic (opcode) correct (AND)

+ 0.5 pts operands correct (R6, R1, R7)

Row 3

+ 0.25 pts Mnemonic (opcode) correct (NOT)

+ 0.5 pts operands correct (R7, R1)

Row 4

+ 0.25 pts Mnemonic (opcode) correct (AND)

+ 0.5 pts operand correct (R5, R0, R7)

Row 5

+ 0.25 pts Mnemonic (opcode) correct (NOT)

+ 0.5 pts operand correct (R5, R5)

Row 6

+ 0.25 pts mnemonic correct (NOT)

+ 0.5 pts operands correct (R6,R6)

Row 7

+ 0.25 pts Mnemonic (opcode) correct (AND)

+ 0.5 pts operands correct (R2, R6, R5)

Row 8

+ 0.25 pts Mnemonic (opcode) correct (NOT)

+ 0.5 pts operands correct (R2, R2)

+ 0 pts incorrect

QUESTION 13

13 2b **5.5 / 9**

+ 9 pts Fully correct

![Screenshot_2023-10-22_at_4.48.11_PM.png](/files/ca55f76e-6b0b-4e6f-9190-8eeaaf3c7d62)

✓ **+ 1 pts** *R0 correct (x8888)*

✓ **+ 1 pts** *R1 correct (xFFFF)*

+ 3 pts R2 correct (x7777)

✓ **+ 1 pts** *R3 correct (x2110)*

✓ **+ 1 pts** *R4 correct (xDEAD)*

✓ **+ 1 pts** *R5 correct (xFFFF)*

+ 0.5 pts R6 correct (x8888)

✓ **+ 0.5 pts** *R7 correct (x0000)*

+ 0 pts Fully Incorrect

QUESTION 14

14 2c **2 / 6**

+ 6 pts Fully Correct (XOR)

✓ **+ 2 pts** *Used OR*

+ 2 pts Incorrect

QUESTION 15

15 2d **2 / 4**

+ 4 pts Correct (40)

✓ **+ 2 pts** *answer is multiple of 5*

+ 2 pts 12 (only counted fetch + decode once for 8 instructions)

+ 2 pts 32 (fetch + decode by no execute)

+ 0 pts incorrect

QUESTION 16

16 3a **7 / 9**

+ 9 pts Fully Correct

JUMPO0 (first row)

+ 6 pts Fully correct

✓ **+ 1 pts** *ADDR1MUX correct (1)*

✓ **+ 1 pts** *ADDR2MUX[1] correct (0)*

✓ **+ 1 pts** *ADDR2MUX[0] correct (1)*

+ 1 pts PCMUX[1] correct (0)

+ 1 pts PCMUX[0] correct (1)

✓ **+ 1 pts** *LD.PC correct (1)*

✓ **+ 1.5 pts** *JUMPO1 (second row) -- blank row*

✓ **+ 1.5 pts** *JUMPO2 (third row) -- blank row*

+ 0 pts Fully incorrect

QUESTION 17

17 3b **3.5 / 3.5**

✓ **+ 1.5 pts** *Correct (yes)*

✓ **+ 2 pts** *correct explanation*

- JMPO does BR + sext6 and because JMP's last 6 bits are 0s it is equivalent to BR + 0

+ 0 pts incorrect

QUESTION 18

18 3c **9 / 9**

✓ **+ 9 pts** *Fully Correct*

XOR0 (first row)

+ 6 pts Fully correct

+ 1.5 pts ALUK[1] correct (1)

+ 1.5 pts ALUK[0] correct (0)

+ 1.5 pts LD.REG correct (1)

+ 1.5 pts GateALU correct (1)

+ 1.5 pts XOR1 (second row) -- blank row

+ 1.5 pts XOR2 (third row) -- blank row

+ 0 pts Fully incorrect

QUESTION 19

19 3d **3.5 / 3.5**

✓ **+ 2.5 pts** *Correct (yes)*

✓ **+ 1 pts** *correct explanation*

- the imm5 and bit 5 of NOT are all 1s , so it would be encoded as DR = SR1 XOR -1 which is flips the bits (NOT SR1)

+ 0 pts incorrect

Your Initials: AP

Name [PRINT CLEARLY]: AARYAN POTDAR

GT username (e.g. gburdell3): apotdar31

CS 2110: Computer Organization and Programming

Gupta/Conte/Adams Fall 2023

QUIZ 3

VERSION C

~~This exam is given under the Georgia Tech Honor Code System.~~ Anyone found to have submitted copied work instead of original work will be dealt with in full accordance with Institute policies.

Georgia Tech Honor Pledge: "*I commit to uphold the ideals of honor and integrity by refusing to betray the trust bestowed upon me as a member of the Georgia Tech community.*"

[MUST sign:] 

- THIS IS A CLOSED BOOK, CLOSED NOTES EXAM
- NO CALCULATORS
- This examination handout has **8** pages.
- Do all your work in this examination handout.
- Only the front of exams sheets will be scanned. Do **not** write your answer on the back of the exam sheets.
- Please write your initials at the top of each page
- WHERE NEEDED, SHOW ALL YOUR INTERMEDIATE RESULTS TO RECEIVE FULL CREDIT

In case you forgot, here are some good facts to know:

Hex	Dec
0x1	1
0x2	2
0x3	3
0x4	4
0x5	5
0x6	6
0x7	7
0x8	8
0x9	9
0xA	10
0xB	11
0xC	12
0xD	13
0xE	14
0xF	15

x	2^x
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16,384
15	32,768
16	65,536

Problem	Points	Score
1	50	
2	25	
3	25	
TOTAL	100	

GOOD LUCK!

More good facts to know:

$1K = 2^{10}$
$1M = 2^{20}$
$1G = 2^{30}$
$1T = 2^{40}$
$1P = 2^{50}$
$1E = 2^{60}$

Your Initials: AP

1. [50 pts] Answer the following short questions. Show your work (where needed) to receive full credit.

→ current, PC* = 0x6401

(a) Imagine there is an LD instruction at address 0x6400. Calculate the following and give your **answer in binary** (**show your work!**):

Note: Accounting for PC* (PC+1)

The PCoffset9 such that the LD will read from address 0x64B4: 0 1011 0100

PC+1 ← PC* + PC offset+9

↳ x6401 → 0110 0100 0000 0001 + 1011 0100 = 0110 0100 1011 0100

The PCoffset9 such that the LD will read from address 0x63FA: 1 1111 1001

PC+1 = x6401 → 0110 0100 0000 0001

0110 0011 1111 1010

0000 0000 0000 0111

0110 0100 0000 0001

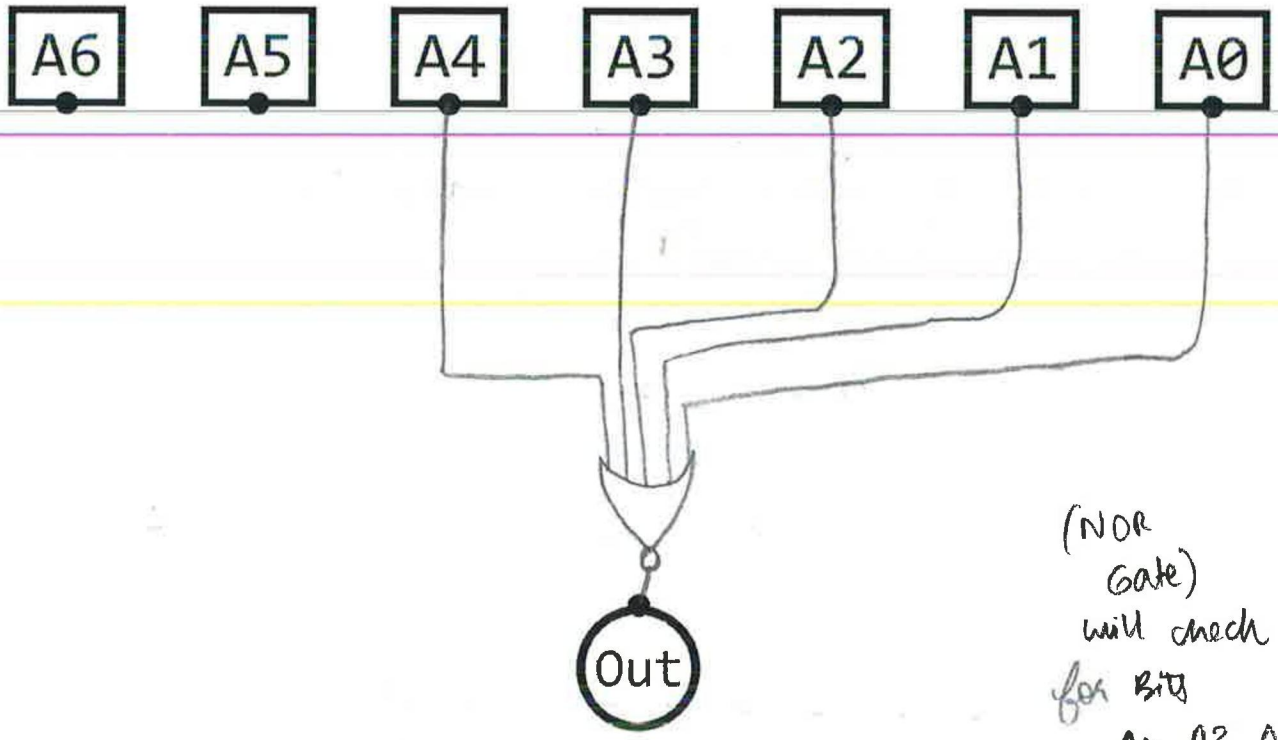
Offset needed is | Negate

0 0000 0111

1 1111 1000 flip

1 1111 1001 +1

(b) Using **only NOR gates**, draw a circuit below that **outputs 1 if and only if a 7-bit input A is divisible by 32 (is a multiple of 32).** (You may assume A is interpreted as unsigned binary. Any number of inputs to a NOR is acceptable. Bit 6 is the most significant bit, and bit 0 is the least significant bit. It is okay not to use all inputs.)

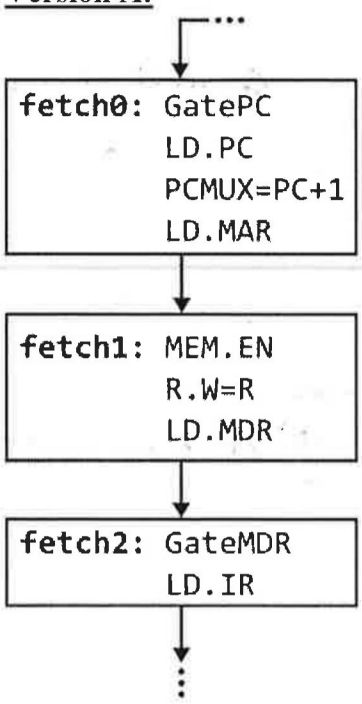


(NOR Gate) will check for Bits A4 A3 A2 A1 & A0 and o/p 1 only if all bits are 0

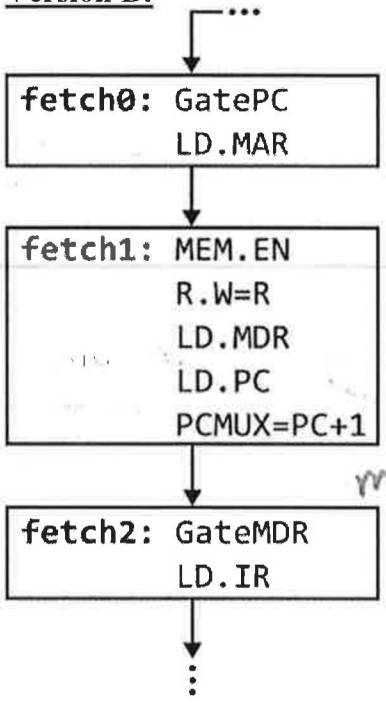
Your Initials: AB

(c) Which of the following, if any, is a valid implementation of the first three microstates of Instruction Fetch for the LC-3? (You may have seen R.W written as MEM.RW in class.)

Version A:



Version B:



→ Not possible must wait for memR

Circle your answer: Only A is valid / Only B is valid / Both are valid / Neither is valid

Explain why:

Version A is valid as PC get simultaneously incremented in fetch 0. The contents of mem (PC) are read & written to IR via LD.IR. Version B is not valid as we must wait for memory Read (R). Hence we cannot simultaneously call PC increment

(d) Write one instruction (encoded as 16 bits) that doubles the number stored in R2 (i.e., performs R2=R2*2): → R2 ← R2 + R2

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x3000	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0

(e) Write two instructions that use JMP and LEA to perform an infinite loop:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x3000	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1
0x3001	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

LEA
JMP

−1 → 0000000001 → 1
1111111110
+ 1 → −1

Your Initials: AP

(f) Answer the following true/false questions by circling "true" or "false," and then give a reason for each answer:

TRUE or (FALSE)	In the LC-3, components such as the PC, IR, MAR, and MDR are level-triggered storage elements. Why or why not? true components in LC3 are edge-triggered so they can be controlled by a clock (0 or 1) & be synchronised.
TRUE or (FALSE)	Von Neumann architectures place instructions and program data in separate memories. Why or why not? Memory holds both data & instructions in the same memory. This is efficient as per VON neuman architecture.
(TRUE) or FALSE	There must be separate microstates in the LC-3 finite state machine (in the control unit) for handling ADD with an immediate operand (e.g., ADD with DR=R1, SR1=R2, and SR2=R3) and ADD with only register operands (e.g., ADD with DR=R1, SR1=R2, and imm5=3). Why or why not? Yes. ADD with Regs and ADD with imm5 will require different signals / selector bits being passed to SR_2MUX (which selects b/w SR2OUT & SEXT(IR[4:0])).
TRUE or (FALSE)	It is useful to assert multiple Gate___ signals (e.g., both GateMDR and GatePC) at once in the LC-3. Why or why not? We can only have one word/value on the bus at a time. Gates put value onto bus. Hence, only one can be asserted at a time / in a clock cycle.
(TRUE)	The following instruction will leave the value in R1 unchanged: 0101001001111111. Why or why not? sent AND imm5. AND-ing with (imm5 = 1111) will set the value in DR(R1) to 1111 1111 1111 1111 → -1 value, therefore changing the value of R1
TRUE or (FALSE)	

Reason →

0101 001 001 imm5

AND DR SR1 11111

(R1)

sext(imm5) = 1111 1111 1111 1111

AND-ing with 1111 1111 1111 1111 does not change the value as A and 1 is A as per logic. Hence, value in R1 is unchanged.

Your Initials: AP

2. [25 pts] Answer the following questions about the following LC-3 program. **Show your work.**

(a) Decode each instruction below and write down the **mnemonic** from the opcode (e.g. LDR) in the **Mnemonic** column. Then write the values of the fields for the instruction in the appropriate columns. For example:
ADD | DR = R1 | SR1 = R2 | SR2 = R3
AND | DR = R1 | SR1 = R2 | imm5 = -7
*If an instruction does not specify one of the fields, leave the cell blank

Address	Value	Mnemonic	DR	SR/SR1	SR2/imm5
0x6000	1001111000111111	NOT	R7	R0	
0x6001	0101110001000111	AND	R6	R1	R7
0x6002	1001111001111111	NOT	R7	R1	
0x6003	0101101000000111	AND	R5	R0	R7
0x6004	1001101101111111	NOT	R5	R5	
0x6005	1001110110111111	NOT	R6	R6	
0x6006	0101010110000101	AND	R2	R6	R5
0x6007	1001010010111111	NOT	R2	R2	

(b) Assuming the provided initial state on the left below (and PC=0x6000), **fill in the state on the right below after the instructions in part (a) execute**, assuming the CPU halts immediately *after* executing the instruction at address 0x6007:

R7 = NOT(R1)
1111 1111 1111 1111
R7 = 0000 0000 0000 0000

R1 = NOT(R0)
1000 1000 1000 1000
= 0111 0111 0111 0111

R6 = R1 & R7
0x FFFF
0x 7777
0x FFFF
1111 1111 1111 1111

R6 = NOT(R6)
= 0x 0000

R2 = R6 AND R5
R2 = x0000
R2 = NOT(R2)
= xFFFF

R5 = R0 AND R7
1000 1000 1000 1000
0000 0000 0000 0000
x0000

R5 = NOT R(5)
R5 = xFFFF

Initial State			New State	
R0	0x8888	→	R0	0x8888
R1	0xFFFF	→	R1	0xFFFF
R2	0x1234	→	R2	0xFFFF
R3	0x2110	→	R3	0x2110
R4	0xDEAD	→	R4	0xDEAD
R5	0xBEEF	→	R5	0xFFFF
R6	0xCAFE	→	R6	0x0000
R7	0x6004	→	R7	0x0000

Your Initials: AP

(c) In general, what bitwise operation do the instructions in part (a) calculate and put into R2, in terms of R0 and R1? **Fill in the blank below:**

R2 ← R0 OR R1

(all value are set to 1)
A or 1 = 1

(d) Assuming fetching (and decoding) an instruction takes 4 cycles, how many cycles in total will the instruction sequence above take? **Show your work.** → 20

R2 ⇐ R0 OR R1

= NOT (NOT(R0) AND NOT(R1))

NOT (R0) → 1 clock cycle

NOT(R1) → 1 clock cycle

AND($\overline{R0}$, $\overline{R1}$) → 1 clock cycle

NOT ($\overline{R0}$ AND $\overline{R1}$) → 1 clock cycle.

a NOT instruction needs 1 clock cycle
an AND " " 1 clock cycle

4 instructions = [3 (Fetch), 1 (decode), 1 (execute)] ×4

= 5×4 = 20 clock cycles

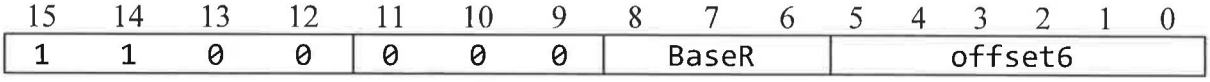
Your Initials: AP

3. [25 pts] Answer the following questions about LC-3 microcode. **Show your work.**

Suppose that the following signals are specified with the following bits:

- ALUK: ADD=00, AND=01, NOT=10, PASSA=11
- ADDR1MUX: PC=0, BaseR=1
- ADDR2MUX: ZERO=00, offset6=01, PCoffset9=10, PCoffset11=11
- PCMUX: PC+1=00, ADDER=01, BUS=10
- MARMUX: ZEXT=0, ADDER=1
- R.W: R=0, W=1

(a) Consider a new LC-3 instruction `JMPO` with the following encoding:



and the following semantics (behavior):

```
PC = BaseR + SEXT(offset6)
```

Fill in the microcode for JMPO below (write 1 bit in each cell, except leave unused rows blank!):

	ADDR1MUX	ADDR2MUX[1]	ADDR2MUX[0]	PCMUX[1]	PCMUX[0]	LD.PC
jmpo0	1	0	1	1	0	1
jmpo1						
jmpo2						

It is okay not to use all rows above. Leave unused rows blank.
Assume any signals not shown are set correctly (including the SR1MUX and DRMUX)

(b) Compare the encoding of JMPO in part (a) with the encoding of JMP (see reference sheet). Can the JMPO microstates you wrote replace the existing JMP microstates in the LC-3 control unit's finite state machine *and* preserve the functionality of the original encoding of JMP? Why or why not?

→ The range of addresses of JMPO is larger than that of JMP (only Base R), hence it can replace JMP. JMPO also needs 1 clock cycle, similar to JMP and BUS used only once. A pc offset = 000000 acts similar to JMP. Hence, the functionality is preserved.

(c) Consider a new LC-3 instruction **XOR** which can be encoded in two forms with different behavior, much like the existing LC-3 **ADD** and **AND** instructions.

The first form is the following:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	DR			SR1			0	0	0	SR2		

which has the following semantics (behavior):

```
DR = SR1 XOR SR2
```

Then there is another form of **XOR** encoded as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	DR			SR1			1	imm5				

which has the following semantics (behavior):

```
DR = SR1 XOR SEXT(imm5)
```

Imagine that ALUK=10 instructs the ALU to perform `A XOR B` instead of `NOT A`. Then fill in the microcode for XOR below (write 1 bit in each cell, except leave unused rows blank!):

	ALUK[1]	ALUK[0]	LD.REG	GateALU
xor0	1	0	1	1
xor1				
xor2				

It is okay not to use all rows above. Leave unused rows blank.
Assume any signals not shown are set correctly
(including the SR1MUX, DR1MUX, and LD.CC signal for condition codes)

(d) Compare the encoding of XOR in part (a) with the encoding of NOT (see reference sheet). Can the XOR microstates you wrote replace the existing NOT microstates in the LC-3 control unit's finite state machine *and* preserve the functionality of the original encoding of NOT? Why or why not?

A	NOT A
1	0
0	1

A	B	A XOR B
1	1	0
0	1	1

→ As seen on the left, A NOT output can be achieved with A XOR B, provided we have the appropriate input value for SR2/imm5. Hence, encoding for XOR can be used to replace NOT as the original functionality can be preserved.