# Quiz 3

Graded

**Student**

[redacted]

**Total Points**

68.5 / 100 pts

**Question 1**

1ai — 2 / 3 pts

+ 3 pts Correct (010100110)

+ 2 pts correct but doesn't use 9 bits (10100110)

+ 1 pt incorrect (off by 1)

✔ + 2 pts correct but not in binary

+ 0 pts incorrect

**Question 2**

1aii — 1 / 3 pts

+ 3 pts Correct (111111000)

+ 2 pts correct but doesn't use 9 bits

+ 2 pts correct but not in binary

✔ + 1 pt incorrect (off by 1)

+ 0 pts incorrect

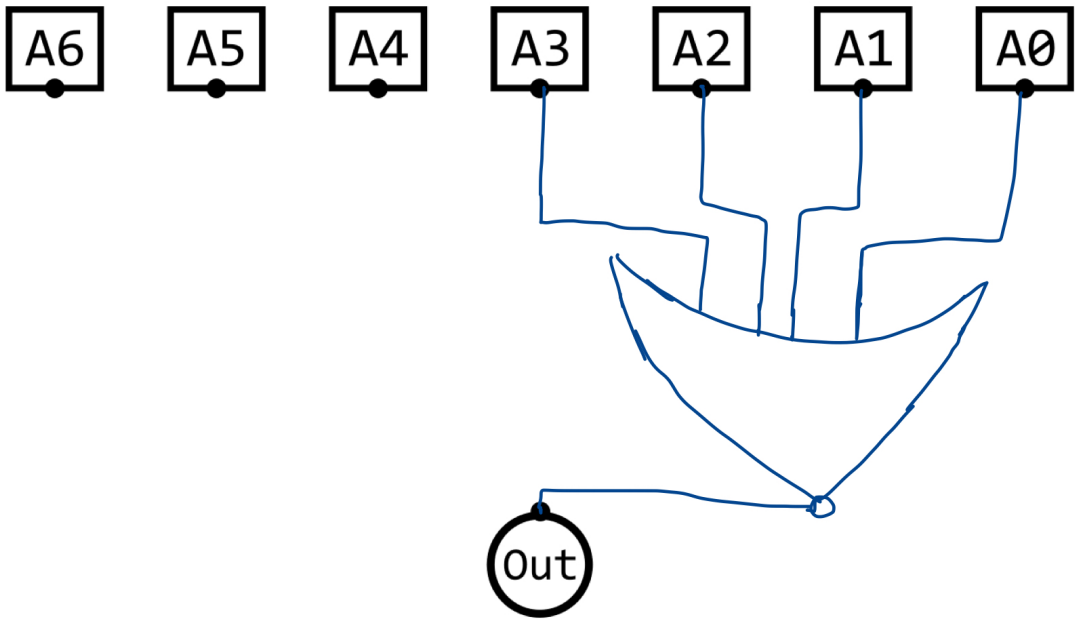
Question 3

1b

Resolved 0 / 5 pts

+ 5 pts Correct

least significant bit. It is okay not to use all inputs.)




+ 4 pts Marked zero as not a multiple of 16

+ 3 pts computed opposite

+ 3 pts Used cascading NORs instead of NOR4

+ 3 pts One input off (extra or missing)

+ 1 pt correct but using gates other than NOR

✔ + 0 pts incorrect

Regrade Request Submitted on: Nov 06

I am only one input extra from the right answer. Are u guys confusing my input bubble as inverters?

An input bubble is an inverter. You cannot use a NOT gate for this question.

Reviewed on: Nov 06

**Question 4**

**1c** **0** / 8 pts

**+ 3 pts** Correct (circled answer)

**+ 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 (selected only Only A, Only B or Neither, or no answer).

**Question 5**

**1d** **6** / 6 pts

✔ **+ 6 pts** ****Correct (0001 1011 0100 0101 aka ADD R5, R5, R5 )

**+ 2 pts** opcode correct ( 0001 aka ADD)

**+ 2 pts** 2/3 registers correct

**– 1 pt** Bit 5 Incorrect

**+ 0 pts** Incorrect

**Question 6**

**1e** **10** / 10 pts

✔ **+ 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**

**1fi** **3** / 3 pts

✔ **+ 1 pt** Correct (True)

✔ **+ 2 pts** explanation correct

› R1 AND all 1s == R1

**+ 0 pts** incorrect

**Question 8**

**1fii** **3** / 3 pts

✔ **+ 1 pt** Correct (False)

✔ **+ 2 pts** explanation correct

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

**+ 0 pts** incorrect

**Question 9**

**1fiii** **3** / 3 pts

✔ **+ 1 pt** Correct (False)

✔ **+ 2 pts** explanation correct

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

**+ 0 pts** incorrect

**Question 10**

**1fiv** **0** / 3 pts

**+ 1 pt** 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 11**

**1fv** **3** / 3 pts

✔ **+ 1 pt** Correct

✔ **+ 2 pts** explanation correct

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

**+ 0 pts** incorrect

**Question 12**

2a **6** / 6 pts

✔ **+ 6 pts** Correct

| | Mnemonic | DR | SR/SR1 | SR2/imm5 |
|---|---|---|---|---|
| | NOT | R2 | R0 | |
| | AND | R4 | R1 | R2 |
| | NOT | R2 | R1 | |
| | AND | R5 | R0 | R2 |
| | NOT | R4 | R4 | |
| | NOT | R5 | R5 | |
| | AND | R3 | R4 | R5 |
| | NOT | R3 | R3 | |

al state on the left below (and PC=0x4000), **fill in the state**
**instructions in part (a) execute**, assuming the CPU halts

Row 1

**+ 0.25 pts** Mnemonic (opcode) correct (NOT)

**+ 0.5 pts** operands correct (R2, R0)

Row 2

**+ 0.25 pts** Mnemonic (opcode) correct (AND)

**+ 0.5 pts** operands correct (R4, R1, R2)

Row 3

**+ 0.25 pts** Mnemonic (opcode) correct (NOT)

**+ 0.5 pts** operands correct (R2, R1)

Row 4

**+ 0.25 pts** Mnemonic (opcode) correct (AND)

+ **0.5 pts** operand correct (R5, R0, R2)

---

Row 5

+ **0.25 pts** Mnemonic (opcode) correct (NOT)

+ **0.5 pts** operand correct (R4, R4)

---

Row 6

+ **0.25 pts** mnemonic correct (NOT)

+ **0.5 pts** operands correct (R5, R5)

---

Row 7

+ **0.25 pts** Mnemonic (opcode) correct (AND)

+ **0.5 pts** operands correct (R3, R4, R5)

---

Row 8

+ **0.25 pts** Mnemonic (opcode) correct (NOT)

+ **0.5 pts** operands correct (R3, R3)

---

+ **0 pts** incorrect

**Question 13**

**2b** **4** / 9 pts

+ **9 pts** Fully correct

✔ + **1 pt** R0 correct (x8888)

✔ + **1 pt** R1 correct (xFFFF)

+ **1 pt** R2 correct (x0000)

+ **3 pts** R3 correct (x7777)

✔ + **1 pt** R4 correct (x8888)

+ **1 pt** R5 correct (xFFFF)

✔ + **0.5 pts** R6 correct (xCAFE)

✔ + **0.5 pts** R7 correct (x0000)

+ **0 pts** Fully Incorrect

**Question 14**

**2c** **2** / 6 pts

+ **6 pts** Fully Correct (XOR)

+ **2 pts** Used OR

✔ + **2 pts** Incorrect

**Question 15**

**2d** **4** / 4 pts

✔ + **4 pts** Correct (40)

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

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

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

+ **0 pts** incorrect

**Question 16**

**3a** **9** / 9 pts

✔ + **9 pts** Fully Correct

---

JUMPO0 (first row)

+ **6 pts** Fully correct

+ **1 pt** ADDR1MUX correct (1)

+ **1 pt** ADDR2MUX[1] correct (0)

+ **1 pt** ADDR2MUX[0] correct (1)

+ **1 pt** PCMUX[1] correct (0)

+ **1 pt** PCMUX[0] correct (1)

+ **1 pt** 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**

**3b** **3.5** / 3.5 pts

✔ **+ 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
- JMPO with offset6 = 0 is equivalent to JMP

**+ 0 pts** incorrect

**Question 18**

**3c** **9** / 9 pts

✔ **+ 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**

**3d** **0** / 3.5 pts

**+ 2.5 pts** Correct (yes)

**+ 1 pt** 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: [redacted]

**Name [PRINT CLEARLY]:** [redacted]

**GT username (e.g. gburdell3):** [redacted]

CS 2110: Computer Organization and Programming
Gupta/Conte/Adams Fall 2023

QUIZ 3
**VERSION A**

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:] [redacted]

- 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: [redacted]

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

(a) Imagine there is an LD instruction at address 0x3200. Calculate the following and give your **answer in binary** (**show your work!**):

0x3201

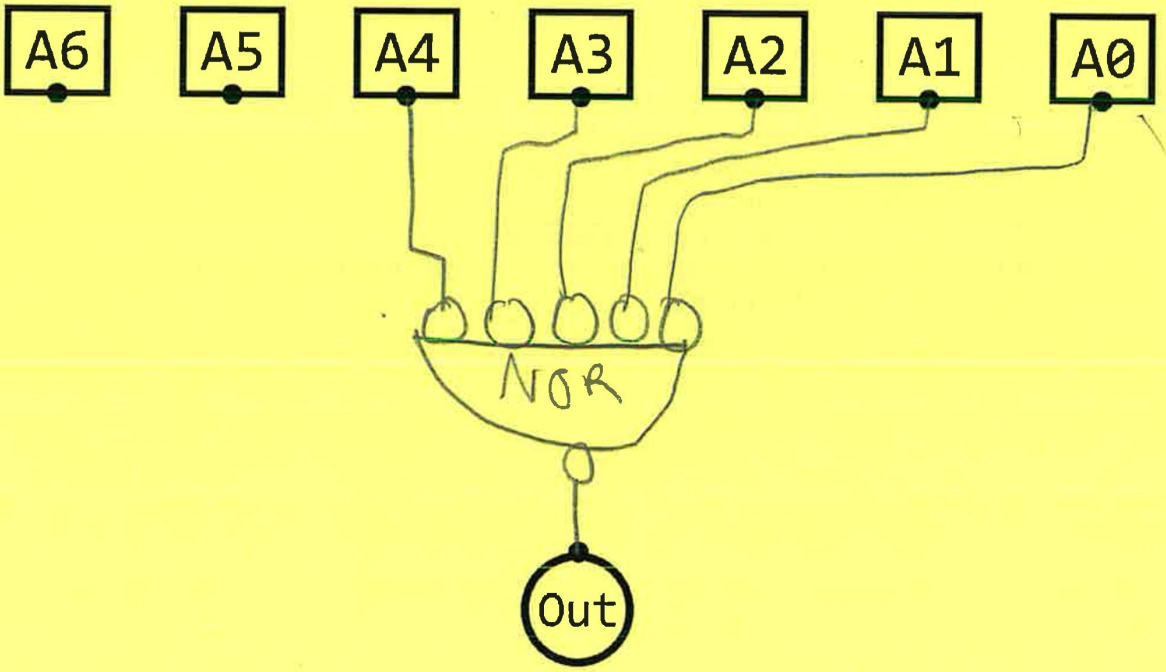
The PCoffset9 such that the LD will read from address 0x32A7: 0x00A6

The PCoffset9 such that the LD will read from address 0x31F9: 0xFFF9

1111 1111 1111 1001

0 1 0 1
1 0 0 0
1 0 0 1

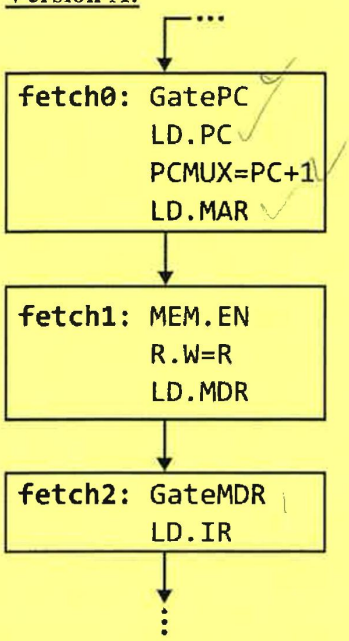
(b) Using **only NOR gates**, draw a circuit below that **outputs 1 if and only if a 7-bit input A is divisible by 16 (is a multiple of 16)**. (You may assume A is interpreted as unsigned binary. Any number of inputs to a NOR is acceptable. Bit 6 is the most significant bit, and bit 0 is the least significant bit. It is okay not to use all inputs.)



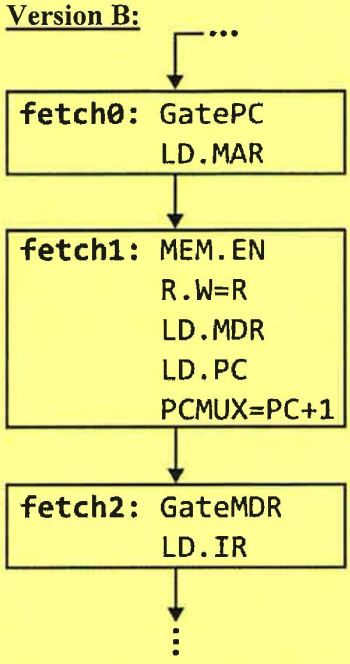
Your Initials: [redacted]

(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:**




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

**Explain why:**

In Fetch, PC is incremented during the first Microstate. Thus A is right and B is Wrong since B increments PC at the second Microstate.

(d) Write one instruction (encoded as 16 bits) that doubles the number stored in R5 (i.e., performs R5=R5*2):

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

(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 |

001
110

Your Initials: [redacted]

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

| | |
|---|---|
| TRUE or FALSE | The following instruction will leave the value in R3 unchanged: 0101011011111111. Why or why not?<br>Yes because this AND instruction will do a bitwise AND against 1, which allows values to remain the same. A AND 1 = A |
| TRUE or FALSE | Von Neumann architectures place instructions and program data in separate memories. Why or why not?<br>No because Data and Instructions are held in 1 memory stage for Von Nuemann. |
| 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?<br>No, because it will cause a short circuit. Only one Gate should be asserted at one time to prevent this. |
| 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?<br>Yes because bit 5 in either case is different. Thus, while Immediate will use the imm5 from the instruction, RegisterAdd will use the register file instead. |
| 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?<br>No, because these are registers controlled by control signals. Registers in LC-3 are edge triggered according to the clock of the computer, |

Your Initials: [redacted]

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 |
|---|---|---|---|---|---|
| 0x4000 | 1001010000111111 | NOT | $R_2$ | $R_0$ | — |
| 0x4001 | 0101100001000010 | AND | $R_4$ | $R_1$ | $R_2$ |
| 0x4002 | 1001010001111111 | NOT | $R_2$ | $R_1$ | — |
| 0x4003 | 0101101000000010 | AND | $R_5$ | $R_0$ | $R_2$ |
| 0x4004 | 1001100100111111 | NOT | $R_4$ | $R_4$ | — |
| 0x4005 | 1001101101111111 | NOT | $R_5$ | $R_5$ | — |
| 0x4006 | 0101011100000101 | AND | $R_3$ | $R_4$ | $R_5$ |
| 0x4007 | 1001011011111111 | NOT | $R_3$ | $R_3$ | — |

(b) Assuming the provided initial state on the left below (and PC=0x4000), **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 0x4007:

$R_2 = NOT\ R_0$
$R_4 = R_1\ AND\ R_2$
$R_2 = NOT\ R_1$
$R_5 = R_0\ AND\ R_2$
$R_4 = NOT\ R_4$
$R_5 = NOT\ R_5$
$R_3 = R_4\ AND\ R_5$
$R_3 = NOT\ R_3$

| | Initial State | | New State |
|---|---|---|---|
| R0 | 0x8888 | → | 0x8888 |
| R1 | 0xFFFF | → | 0xFFFF |
| R2 | 0x1234 | → | 0x1111 |
| R3 | 0x2110 | → | 0x0000 |
| R4 | 0xDEAD | → | 0x8888 |
| R5 | 0xBEEF | → | 0x0000 |
| R6 | 0xCAFE | → | 0xCAFE |
| R7 | 0x0000 | → | 0x0000 |

1000
1111
0000

100
001
000

1000
0111
1000

Your Initials: [redacted]

(c) In general, what bitwise operation do the instructions in part (a) calculate and put into R3, in terms of R0 and R1? **Fill in the blank below:**

R3 ← R0 NOR R1

(d) Assuming fetching (and decoding) an instruction takes 1 cycles, how many cycles in total will the instruction sequence above take? **Show your work.**

NOT = 1 instruction cycle + 4 Instruction Cycles

AND = 1 instruction cycle + 4 Instruction Cycles

5 NOTs + 3 AND = 5(5) + 3(5) = 25 + 15 = 40

Your Initials: [redacted]

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:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | BaseR | | | offset6 | | | | | |

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 | 0 | 1 | 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?

Yes it can because JMP is already advantageous for using a base register value to access all possible 16-bit addresses in a LC-3... Since JMPO does the same, but simply adds on an offset, the range of the function compared to JMP is the same. Everything Preserved.

Your Initials: [redacted]

(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 (c) 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?

No. Although XOR can be used to toggle bits and mimick NOT, the XOR instruction above can only toggle bits [6:0]. This is not effective since a NOT function toggles all 16-bits from a value, not just 7 bits like the above XOR. So No.