Homework 3

T1

Suppose a 32-bit instruction takes the following format:

OPCODE	SR	DR	IMM

If there are 64 opcodes and 56 registers, what is the range of values that can be represented by the immediate (IMM)? Assume IMM is a 2's complement value.

T2

An LC-3 program is stored in memory locations `x3000` to `x3005`. Note that the branch instruction in memory location `x3002` has an unspecified PCoffset9, denoted as **X**.

Address	Instruction
`x3000`	0101 000 000 1 00000
`x3001`	0001 000 000 1 00010
`x3002`	0000 011 **X**
`x3003`	0001 000 000 1 00011
`x3004`	0001 000 000 1 00001
`x3005`	1111 0000 0010 0101

The program starts executing with `PC = x3000`.

Your job: In the table below, for each value of **X**, answer the question: "Does the program halt?" (Yes or No). If your answer is "Yes", answer the question: "What value is stored in `R0` immediately after the instruction at `x3005` completes execution?" If your answer is "No", put a dash in the column labeled "Value stored in `R0`".

X	Does the program halt?	Value stored in `R0`
000000010		
000000001		
000000000		
111111111		
111111110		

T3

After these two instructions execute: The next instruction to execute will be the instruction at `x3009` if what?

Address	Instruction
`x3000`	0001 000 001 0 00 010
`x3001`	0000 011 000000111

T4

Suppose we changed the LC-3 to have only **four** registers instead of 8. Fewer registers is in general a bad idea since it means loading from memory and storing to memory more often, but we can still ask the question: would there be any benefit to reducing the number of registers? For each of the following, answer yes or no, and explain your answer.

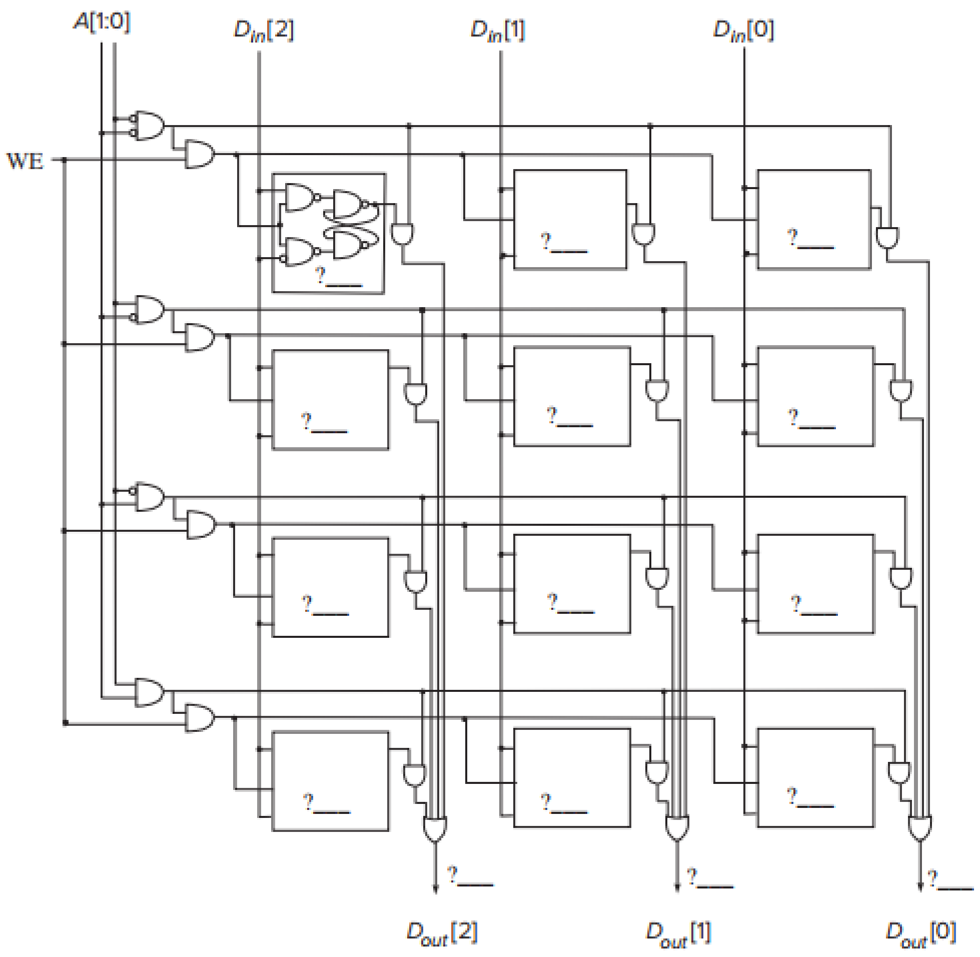
1. If we keep the basic format of all instructions as they currently are (and keep each instruction 16 bits), is there any benefit for operate (0001, 0101, 1001) instructions, if we reduce the number of registers to 4?
2. Is there any benefit for load (0010) and store (0011) instructions, if we reduce the number of registers to 4?
3. Is there any benefit for conditional branch (0000) instructions, if we reduce the number of registers to 4?

T5

We've got a $2^2\text{-by-}3$ bit memory, which is **initialized to store the value 1** in every cell. The table below shows the values of the two-bit address A, one-bit write enable WE, and three-bit datain signals D_{in} during each access:

Cycle No.	A[1:0]	WE	D_{in}[2:0]
1	0 1	1	1 0 1
2	1 1	0	1 1 0
3	1 0	1	0 1 0
4	0 1	1	0 1 1
5	1 1	0	1 0 1
6	0 0	1	1 0 1
7	1 1	1	1 1 0
8	1 1	0	0 1 0

You are required to fill in the values stored in each memory cell and the three data-out lines just **before the end of the eighth cycle** (those marked by question marks `?`).



T6

Consider a memory that we will perform five successive accesses to. The following table shows the type of each access (**R**ead (load), **W**rite (store)), and the contents of the MAR and MDR at the **completion** of the access. Note that we have shortened the addressability to 5 bits.

Operation No.	R/W	MAR	MDR
1	W	x___	11110
2	_	x___	_____
3	W	x___	10___
4	_	x___	_____
5	_	x___	_____

Operations on Memory

The following table show the contents of memory locations at `x4000` to `x4004` *before the first access, after the third access, and after the fifth access*. We have added a constraint to this problem in order to get one correct answer: The MDR can **ONLY** be loaded from memory as a result of a load (read) access.

Address	Before Access 1	After Access 3	After Access 5
`x4000`	01101	____0	_____
`x4001`	11010	_0__0	_____
`x4002`	_1___	_____	_____
`x4003`	10110	_____	01101
`x4004`	11110	11110	11110

Contents of Memory locations

You're required to fill in the blanks.

T7

1. If a machine cycle is 5 nanoseconds (i.e., 5×10^{-9} seconds), how many machine cycles occur each second?
2. Suppose the computer requires an average of **eight cycles** to process each instruction, and the computer processes instructions **one at a time** from beginning to end. Then how many instructions can the computer process in 1 second?
3. Modern microprocessors usually uses the **pipeline** technique to fully utilize the CPU. Pipeline is computer's equivalent of an assembly line. Each phase of the instruction cycle is implemented as one or more separate pieces of logic. Each step in the processing of an instruction picks up where the previous step left off in the previous machine cycle. Using this feature, an instruction can be fetched from memory **every machine cycle** and handed off at the end of the machine cycle to the decoder, which performs the decoding function during the next machine cycle while the next instruction is being fetched. In short, by properly dividing an instruction into multiple phases, we can run several instructions at the same time. How many instructions can the computer process in 1 second if we apply the pipeline technique?

T8

The LC-3 does not have an opcode for the logical function XOR. The eight instruction sequence below performs the XOR of the contents of **R1** and **R2** and puts the result in **R3**. Fill in the four missing instructions so that the eight instruction sequence will do the job.

Address	Instruction
`x3000`	
`x3001`	1001 110 010 111111
`x3002`	0101 101 111 000 010
`x3003`	
`x3004`	1001 001 101 111111
`x3005`	
`x3006`	
`x3007`	1001 011 000 111111

T9

We would like to have an instruction that does nothing. Many ISAs actually have an opcode devoted to doing nothing. It is usually called **NOP**, which means NO OPERATION. The instruction is fetched, decoded, and executed. The execution phase is to do nothing! Which of the following five instructions could be used for NOP and have the program still work correctly? For other instructions, please describe what they have done.

1. 0001 010 001 1 00010
2. 0000 111 000000000
3. 0000 101 000000100
4. 1001 010 111 111111
5. 1111 0000 00100011

T10

Please describe the limitations of the BR instruction in LC-3 and how JMP instruction addresses the issue.