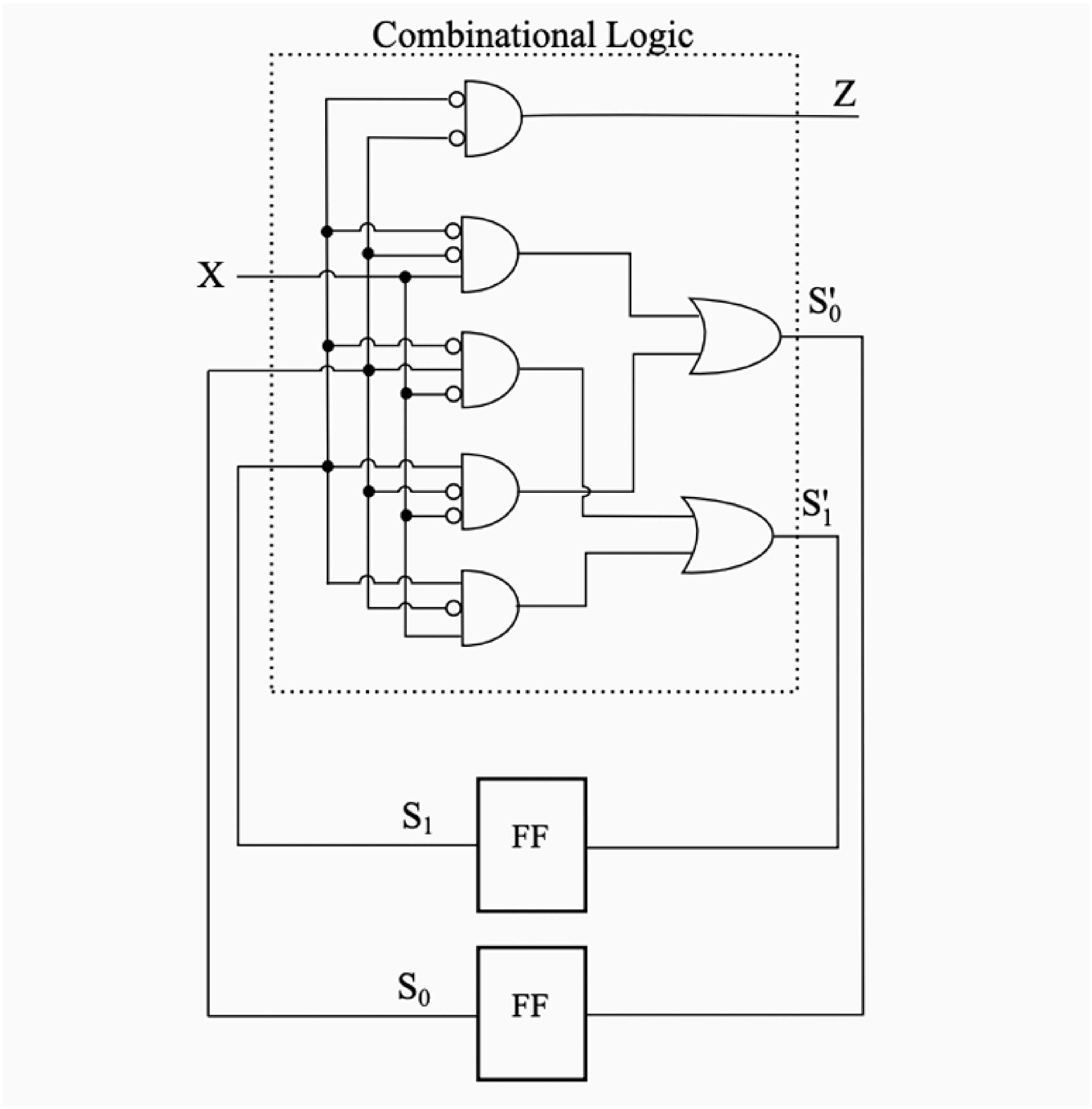
T1

The logic diagram shown below is a finite state machine.



(1). Construct the truth table for the combinational logic

S_1	S_0	X	Z	S_1'	S_0'
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

(2). Draw the state machine.

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 = x4000.

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 x4004 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

Say it takes 100 cycles to read from or write to memory and only one cycle to read from or write to a register.

Calculate the number of cycles it takes for each phase of the instruction cycle for the LC-3 instruction

```
ADD R6, R2, R6
```

Assume each phase (if required) takes one cycle, unless a memory access is required.

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

Suppose a 32-bit instruction takes the following format:

OPCODE	DR	SR1	SR2	UNUSED

If there are 225 opcodes and 120 registers,

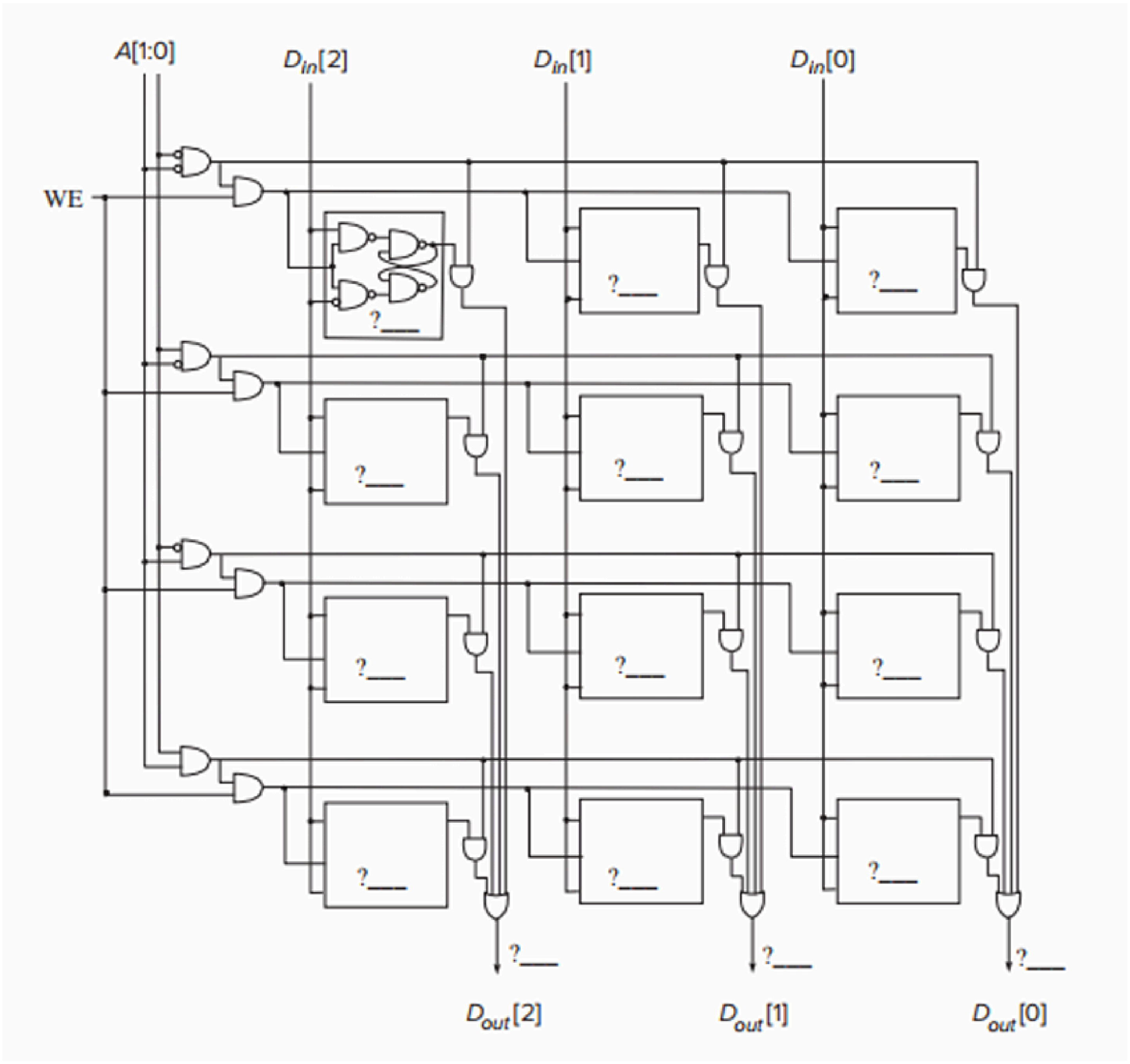
(a). What is the minimum number of bits required to represent the OPCODE?
(b). What is the minimum number of bits required to represent the destination register (DR)?
(c). What is the maximum number of UNUSED bits in the instruction encoding?

T6

The 2^2 by 3 bit memory discussed in class is accessed during five consecutive clock cycles. The table below shows the values of the two-bit address, one-bit write enable, and three-bit data-in signals during each access.

	A[1:0]	WE	D_{in}[2:0]
cycle 1	0 1	1	1 0 1
cycle 2	1 1	0	1 1 0
cycle 3	1 0	1	0 1 0
cycle 4	0 1	1	0 1 1
cycle 5	1 1	0	1 0 1
cycle 6	0 0	1	1 1 1
cycle 7	1 1	1	1 1 1
cycle 8	1 1	0	0 1 0

Your job: Fill in the value stored in each memory cell and the three data-out lines just before the end of the eighth cycle. Assume initially that all 12 memory cells store the value 1. In the figure below, each question mark (?) indicates a value that you need to fill in.



T7

Consider a memory that we will perform five successive accesses to. The following table shows the type of each access (Read (load), Write (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.

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(please give an simple example)