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.

opcode 有 64 = 2^6条，则需要 6bit 才能表示完整。 register 有 56 个，也需要 6bit 才能表示完整。
则 IMM 可用位数为： 32 - 6 - 6 - 6 = 14bit 则表示范围： $-2^{13} < IMM < 2^{13} - 1$

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 AND R0,R0,0
`x3001`	0001 000 000 1 00010 ADD R0,R0,2
`x3002`	0000 011 **X** **BRzp,X**
`x3003`	0001 000 000 1 00011 ADD R0,R0,3
`x3004`	0001 000 000 1 00001 ADD R0,R0,1
`x3005`	1111 0000 0010 0101 HALT

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 `x3004` completes execution?" If your answer is "No", put a dash in the column labeled "Value stored in `R0`".

前 4 个都比较显然，第 5 个注意循环 `R0` 循环加 2 直到加到 x8000后循环中止(识别为负)，再加上后面的 3 和 1，得到 x8004.

X	Does the program halt?	Value stored in `R0`
000000010	Yes(跳至x3005)	2
000000001	Yes(跳至x3004)	3
000000000	Yes(跳至x3003)	6
111111111	No(跳至x3002)	--
111111110	Yes(跳至x3001)	x8004 ($-2^{15}+4$)

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 ADD R0,R1,R2
`x3001`	0000 011 000000111 BRzp,7

当 `R1` 加 `R2` 的和为正数或 0 时，之后执行到 x3039.

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?

   ADD(0001) 和 AND(0101) 有了更大的立即数范围。但 NOT(1001) 没有获益。

2. Is there any benefit for load (0010) and store (0011) instructions, if we reduce the number of registers to 4?

   LD(0010) 和 ST(0011) 能有更多的一位去寻址。

3. Is there any benefit for conditional branch (0000) instructions, if we reduce the number of registers to 4?

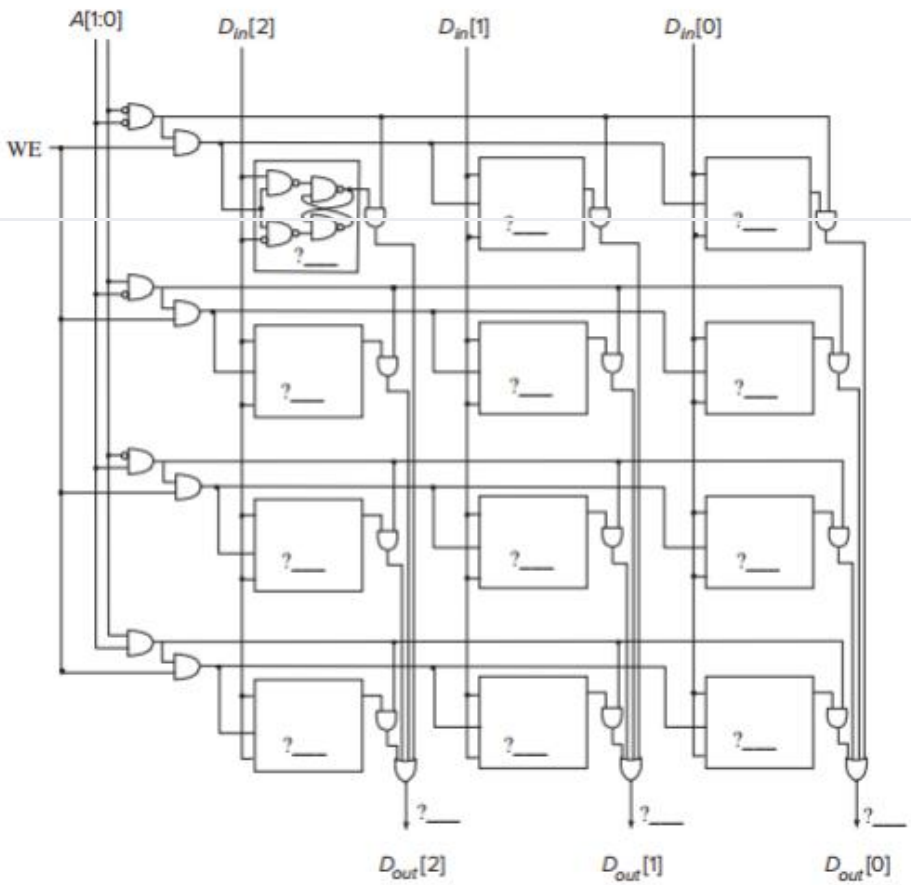
   BR(0000) 中没有寄存器，因此没有获益。

T5

We've got a 2^2-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 data-in 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 ？).



101, 011, 010, 110, 110

111	101	111	111	101
111	101	111	111	111
111	101	010	111	010
111	011	010	111	011
111	011	010	111	111
101	011	010	111	101
101	011	010	110	110
101	011	010	110	110

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	x4000(4)	11110
2	R(1)	X4003(3)	10110(3)
3	W	X4001(4)	10110(3)
4	R(5)	X4002(5)	01101(5)
5	W(5)	X4003(5)	01101(5)

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	11110(4)	11110(5)
`x4001`	11010	10110(4)	10110(5)
`x4002`	01101(5)	01101(5)	01101(5)
`x4003`	10110	10110(2)	01101
`x4004`	11110	11110	11110

Contents of Memory locations

You're required to fill in the blanks.

1: 1,3的mdr改变->一定有一次读,2R
2: 1,2,3两次W,一次R,x4000,x4001改变，所以其他的不变
3: 而3的mdr一定没改变,所以来自于2的R，找寻相似数据，只有x4003:10110,填回
4: 结合3，易知3W改变的是x4001,所以1W改变x4000,并且可以填好after3的部分内存
5: 观察得知，x4003内存在45之后改变，且值不等于3的mdr，所以一定是先R再W，找寻after3之后值为01101的内存，发现没有，所以只可能是x4002，且除了x4003不会有内存改变

Operation No.	R/W	MAR	MDR
1	W	x4000	11110
2	R	x4003	10110
3	W	x4001	10110
4	R	x4002	01101
5	W	x4003	01101

Operations on Memory

Address	Before Access 1	After Access 3	After Access 5
x4000	01101	11110	11110
x4001	11010	10110	10110
x4002	01101	01101	01101
x4003	10110	10110	01101
x4004	11110	11110	11110

Contents of Memory locations

T7

1. If a machine cycle is 5 nanoseconds (i.e., 5×10^{-9} seconds), how many machine cycles occur each second?

   $1/(5*10^{-9})=2*10^{8}$

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?

   $2*10^{8}/8=2.5 \times 10^{7}$

3. Modern microprocessors usually uses the **pipeline(流水线)** 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?相当于每个机器周期执行一条指令2×10^{8}

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 NOT R6,R2
x3002	0101 101 111 000 010 AND R5,R7,R2
x3003	
x3004	1001 001 101 111111 NOT R1,R5
x3005	
x3006	
x3007	1001 011 000 111111 NOT R3,R0

以下用 $a \oplus b$ 表示 a XOR b，$a + b$ 表示 a OR b，$a \cdot b$ 表示 a AND b，$\bar{a}$ 表示 NOT a，则有 $a \oplus b = a \cdot \bar{b} + \bar{a} \cdot b = a \cdot \bar{b} + \bar{a} \cdot b = \overline{\bar{a} \cdot \bar{b}} \cdot \overline{a \cdot b}$.

Address	Instruction	Comments
x3000	**1001 111 001 111111**	NOT R7, R1
x3001	1001 110 010 111111	NOT R6, R2
x3002	0101 101 111 000 010	AND R5, R7, R2
x3003	**0101 100 110 000 001**	AND R4, R6, R1
x3004	1001 001 101 111111	NOT R1, R5
x3005	**1001 010 100 111111**	NOT R2, R4
x3006	**0101 000 001 000 010**	AND R0, R1, R2
x3007	1001 011 000 111111	NOT R3, R0

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 将R1中的值与立即数2相加并存放到R2中
2. 0000 111 000000000 NOP
3. 0000 101 000000100 检查标志位N和P，若其中一个被置位则跳转到对应的位置
4. 1001 010 111 111111对R7求反存放到R2中
5. 1111 0000 00100011 TRAP指令，读取内存单元x0023中的内容作为服务程序入口

T10

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

BR指令无法跳转到PC寄存器和偏移量之和范围以外的指令，即其最大范围为距离当前指令+256到-255的地址空间

对于JMP指令，其能够将对应寄存器的内容装入PC寄存器，使得程序执行流可以跳转至内存空间的任意位置。