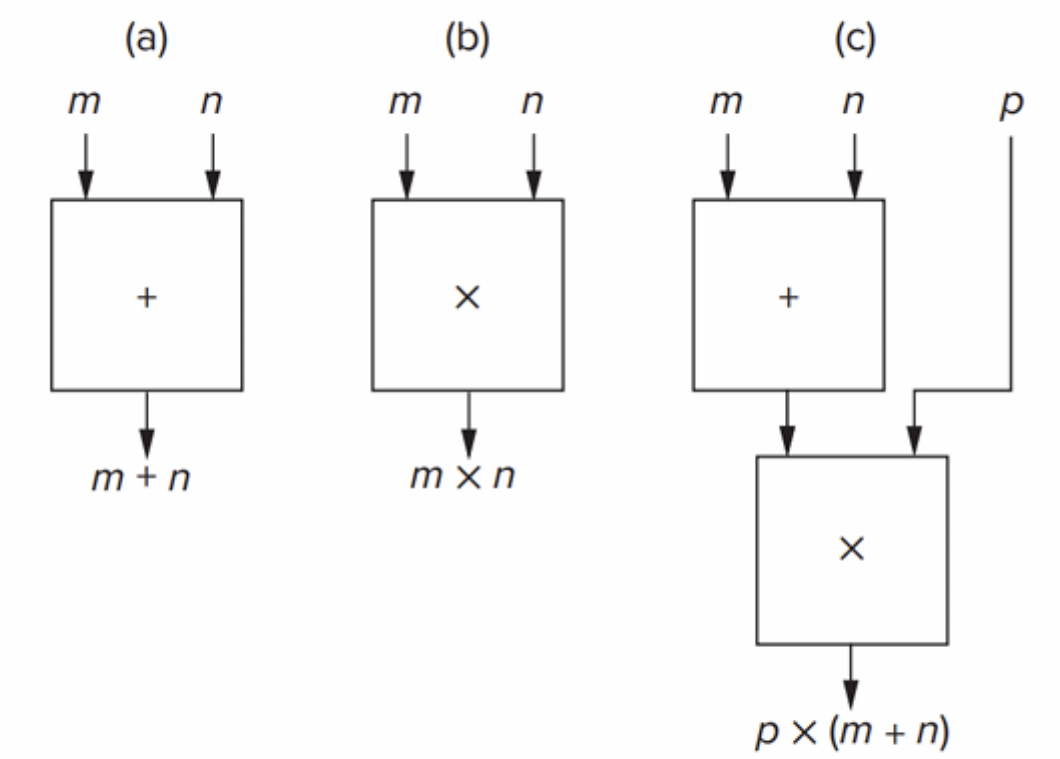
ICS 习题课

作业1

Say we had a "black box," which takes two numbers as input and outputs their sum. See Figure 1.10a.
Say we had another box capable of multiplying two numbers together. See Figure 1.10b.
We can connect these boxes together to calculate p × (m + n). See Figure 1.10c.

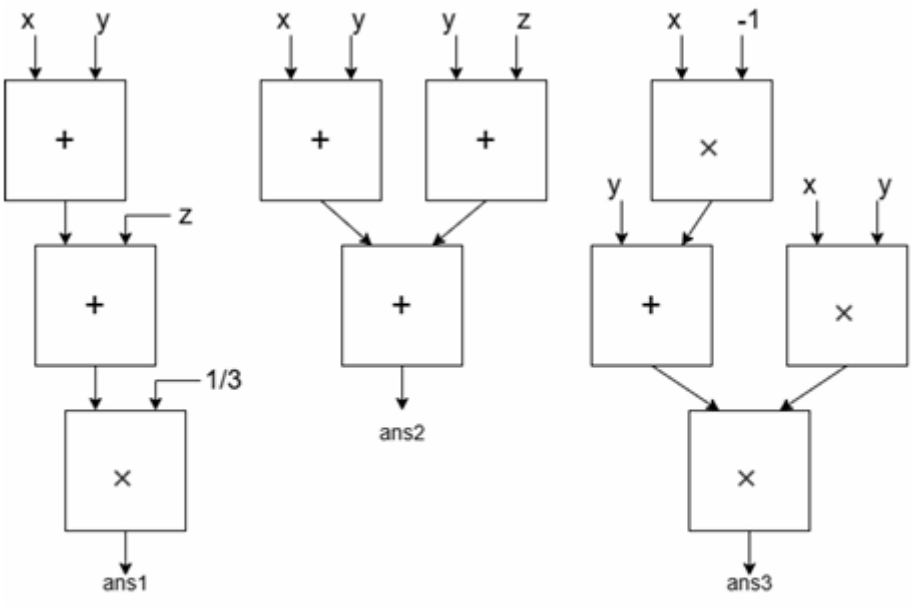
T1

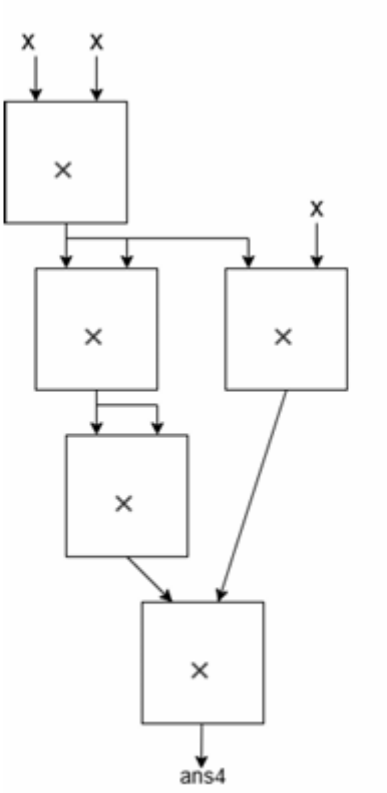


Assume we have an unlimited number of these boxes. Show how to connect them together to calculate:

1. The average of the three input numbers x, y, and z.
2. x+2y+z
3. $xy^2 - x^2y$
4. How many boxes do you need at least to calculate x^{11}?

这道题比较简单


x
y
+
z
+
1/3
×
ans1
x
y
y
z
+
+
+
ans2
x
-1
×
y
x
y
+
×
×
ans3


x
x
×
x
×
×
×
×
ans4

Convert these decimal numbers to 8-bit 2's complement numbers:

1. 73
2. 46
3. -115

T2

Convert the following 8-bit 2's complement numbers to decimal:

1. 0101 1010
2. 1111 0110
3. 1000 0110

> 基础题，注意不要算错了，不放心可以反过来验证一下
>
> (1) 0100 1001 / 0010 1110 / 1000 1101
>
> (2) 90 / -10 / -122

T3

Compute the following and answer it in decimal. Assume each operand is a 2's complement binary number.

1. 11 + 01010101
2. 01001 - 111010
3. 1010 − 01011

> 先把数字转换成十进制 直接计算
>
> (1) -1 + 85 = 84
>
> (2) 9 - (-6) = 15
>
> (3) -6 - 11 = -17

T4

Write your result in binary and decimal.

1. What is the largest positive number one can represent in an eight-bit 2's complement code?
2. What is the greatest magnitude negative number one can represent in an eight-bit 2's complement code?
3. What is the largest positive number one can represent in n-bit 2's complement code?
4. What is the negative number with the largest absolute value one can represent in n-bit 2's complement code?

> (1) 八位 2 的补码能表示的最大正数: 0111 11111 / 127
>
> (2) 八位 2 的补码能表示的最大(绝对值)负数: 1000 0000 / -128
>
> (3) n 位 2 的补码能表示的最大正数: 011 ... 1111 / $2^{n-1}-1$
>
> (4) n 位 2 的补码能表示的最大(绝对值)负数: 100 ... 0000 / -2^{n-1}

T5

Describe what conditions indicate overflow has occurred when two 2's complement numbers are added?
Describe what conditions indicate overflow has occurred when two unsigned numbers are added?

> 两个 2 的补码表示的数字相加 会在什么情况下溢出:
>
> 对于 n 位的 2 的补码表示的数字，正数+负数 不会溢出；两个正数或者两个负数相加，两者相加的绝对值 大于 2^{n-1} 时会发生溢出
>
> 两个无符号数字相加 会在什么情况下溢出:
>
> 对于 n 位的无符号的 2 进制表示的数字，两者相加的绝对值大于 2^n 时会发生溢出

Write the decimal equivalents for the following IEEE floating point numbers:

1. `0 10010010 01110000000000000000000`
2. `1 00001110 10011000000000000000000`

T6

Write IEEE floating point representation of the following decimal numbers:

1. 5.375
2. $-10\frac{9}{32}$

(1) 写出下列 IEEE 表示的浮点数的 十进制表示：

- $(-1)^0 \times (1+0.4375) \times 2^{146-127} = 1.4375 \times 2^{19}$
- $(-1)^1 \times (1+0.59375) \times 2^{14-127} = -1.59375 \times 2^{-113}$

(2) 写出下列十进制数字的 IEEE 浮点数

- $5.375 = (101.011)_2 = 1.01011 \times 2^2 = (-1)^0 \times (1+0.01011) \times 2^{129-127}$
  $= 0\ 10000001\ 01011000000000000000000$
- $-10\frac{9}{32} = (1010.01001)_2 = 1.01001001 \times 2^3 = (-1)^1 \times (1+0.01001001) \times 2^{130-127}$
  $= 1\ 10000010\ 01001001000000000000000$

T7

What are the largest and smallest exponents the IEEE standard allows for a 32-bit floating point number? (Answer in decimal)

What about the smallest number regardless of infinity? And the smallest positive number? (Answer in binary)

- 阶码全0：
  - 尾数全为0 表示的数字位0
  - 尾数不全为0 取消隐含1的规则
- 阶码全1：
  - 尾数全为0 代表无穷
  - 尾数不全为0 无效数字

IEEE 表示的32位浮点数 的最大最小指数部分：

- 最大指数部分： 1111 1110 / 254-127=127
- 最小指数部分： 0000 0001 / 1-127=-126

不考虑无限的最小的数字和最小的正数：

- 最小数字： $1\ 11111110\ 11111111111111111111111 = -(2-2^{-23}) \times 2^{127}$
- 最小的正数：

- Normalized: $0\ 00000001\ 00000000000000000000000 = 1.0 \times 2^{-126}$
- Subnormal: $0\ 00000000\ 00000000000000000000001 = 1.0 \times 2^{-126} \times 2^{-23}$

T8

Compute the following and answer in **hexadecimal**:

1. (0011 AND 0110) AND 1101
2. 0101 0111 OR NOT(1101 0111)
3. (1101 0010 OR 0001 1001) OR NOT(0110 1101 AND 1010 1110)

What strategy would you use to design a program that can quickly compute the result of a long series of n-bit AND operations?

计算下列表达式并用十六进制写出答案

0x0 / 0x7F / 0xDB

如何设计一个程序能够快速计算 n 位的 与 操作

言之有理即可：每一位并行计算

T9

Refer to Example 2.11(Page 43) for the following questions.

1. What mask value and what operation would one use to indicate that machine 2 is busy?
2. What mask value and what operation would one use to indicate that machines 2 and 6 are no longer busy?
3. What mask value and what operation would one use to indicate that all machines are busy?
4. What mask value and what operation would one use to indicate that all machines are idle?
5. Using the operations discussed in this chapter, develop a procedure to isolate the status bit of **machine 5** as the sign bit. For example, if the BUSYNESS pattern is 01**0**11100, then the output of this procedure is 00000000. If the BUSYNESS pattern is 01**1**10011, then the output is 10000000. ***Hint:*** What happens when you ADD a bit pattern to itself?

对于 8 位 2 进制 将某个机器设置为busy 即为 在不操作其他位的情况下将对应位变成0； 将某个机器设置为 idle 即为 在不操作其他位的情况下将对应位变成1

- 如何表示机器2 忙： AND 1111 1011
- 如何表示机器2和6不再忙： OR 0100 0100
- 如何表示所有机器忙：AND 0000 0000
- 如何表示所有机器空闲： OR 1111 11111
- 如何隔离处机器5：思路先提取位数然后左移
  - 先 AND 0010 0000

- 连续自己加自己(相当于左移)

T10

Fill in the truth table for the equations given.

```
Q1 = NOT(X AND Z) AND (X AND Y OR Z)
Q2 = NOT(Y OR Z) AND NOT(X AND Y AND Z)
```

X	Y	Z	Q1	Q2
0	0	0	0	1
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	0	0

作业2

T1

Prove that the NOR gate, by itself, is logically complete (see Section 3.3.5) by constructing a logic circuit that performs the AND function, a logic circuit that performs the NOT function, and a logic circuit that performs the OR function. Use only NAND gates in these three logic circuits.

证明 NOR 的完备性，即证明 NOR 可以表示与或非逻辑门

$$NOT(A) = \overline{A} = \overline{A + A} = NOR(A, A)$$

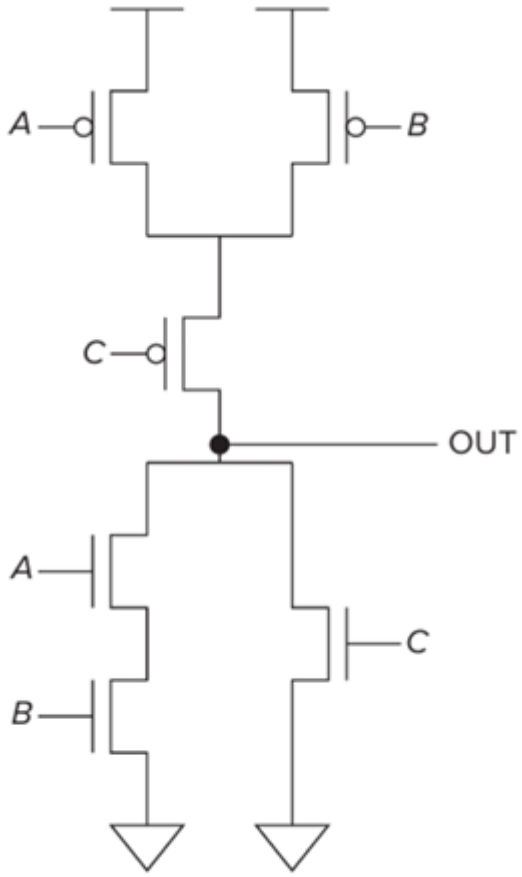
$$AND(A, B) = A \cdot B = \overline{\overline{A} + \overline{B}} = NOR(NOT(A), NOT(B))$$

$$OR(A, B) = A + B = \overline{\overline{A + B}} = NOT(NOR(A, B))$$

逻辑电路略

T2

Transistor circuit shown below produces the accompanying truth table. The inputs to some of the gates of the transistors are not specified. Also, the outputs for some of the input combinations of the truth table are not specified. Complete both specifications. i.e., all transistors will have their gates properly labeled with either A, B, or C, and all rows of the truth table will have a 0 or 1 specified as the output.



A	B	C		OUT
0	0	0		1
0	0	1		0
0	1	0		1
0	1	1		0
1	0	0		1
1	0	1		0
1	1	0		0
1	1	1		0

T3

Shown below are several logical identities with one item missing in each. X represents the case where it can be replaced by either a 0 or a 1 and the identity will still hold. Your job: Fill in the blanks with either a 0, 1, or X. For example, in part a, the missing item is X. That is 0 OR 0 = 0 and 0 OR 1 = 1.

0 OR X = X

1 OR X = 1

0 AND X = 0

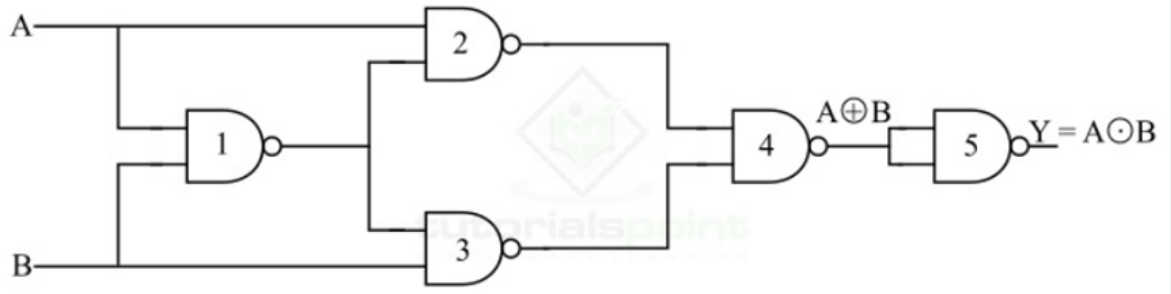
1 AND X = X

0 XOR X = X

T4

Design a XNOR gate with NAND gates.

A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1



先表示出其余的逻辑门

$$NOT(A) = \overline{AA} = NAND(A, A)$$

$$AND(A, B) = \overline{\overline{AB}} = NOT(NAND(A, B))$$

$$ADD(A, B) = \overline{\overline{A + B}} = \overline{\overline{A} \cdot \overline{B}} = NAND(NOT(A), NOT(B))$$

然后自由操作

T5

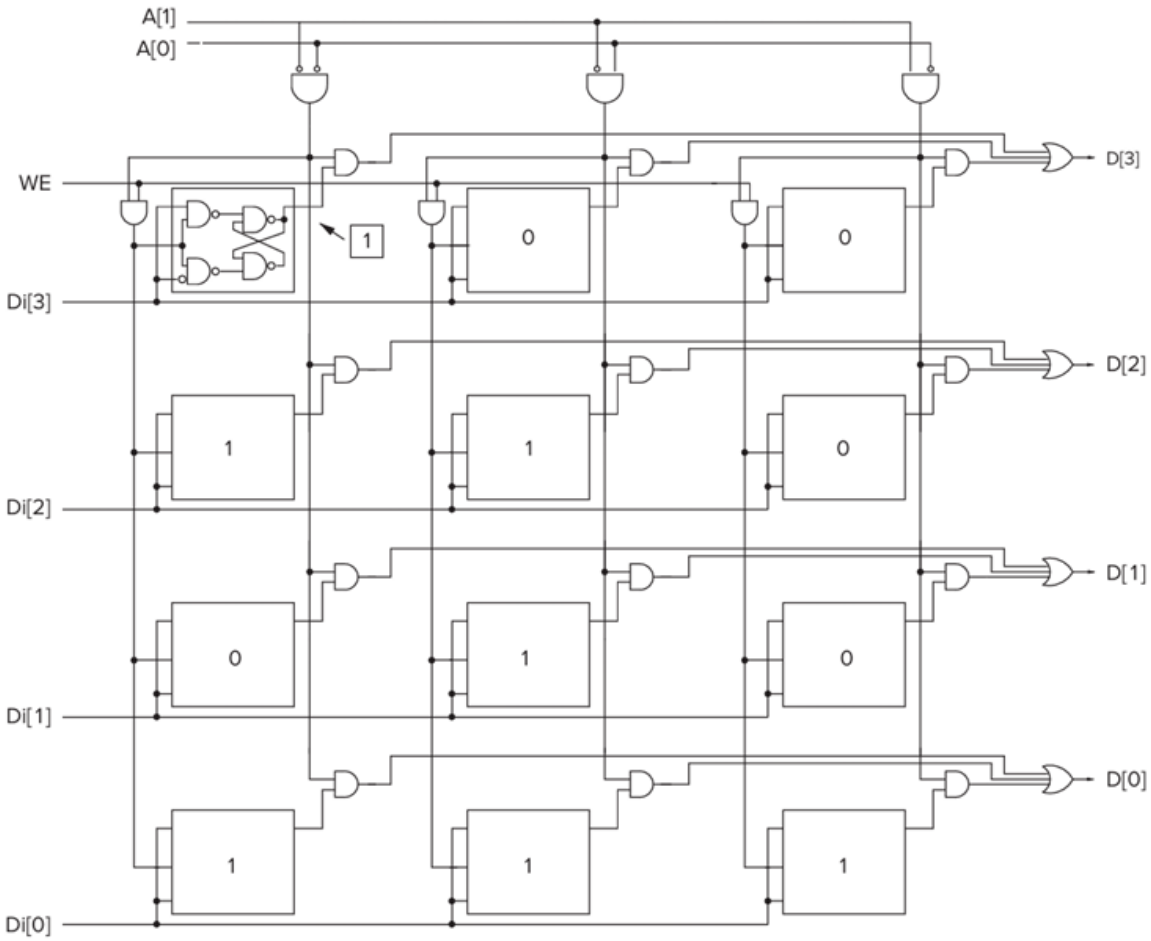


Figure 3.45 Diagram for Exercise 3.40.

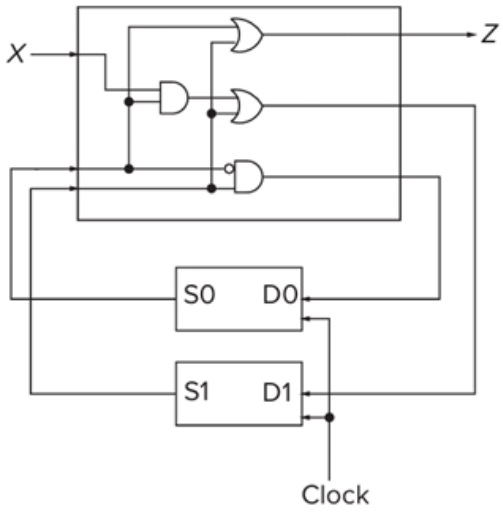
- What is the address space?
- What is the addressability?
- What is the data at address 1?

2. If a computer has eight-byte addressability and needs four bits to access a location in memory, what is the total size of memory in bytes?

> (1) 根据图片回答、
>
> Di是用来写的数据 D是用来读的数据 A是地址控制 WE是读写控制
>
> - 地址空间：3 (存了三个数据)
> - 寻址能力：4 (一个数据四位)
> - address 1 处的数据：0111 (address 01处存的值)
>
> (2) 如果一个计算机 8-byte 寻址能力 同时需要四位访存，整个内存的空间：
>
> 4位表示的内存空间大小 $2^4 = 16$，一个内存地址处是 8 个字节
>
> 内存空间大小 16x8 = 128 字节

T6

The following figure shows an implementation of a finite state machine with an input X and output Z. S1, S0 specifies the present state. D1, D0 specifies the next state.



1. Complete the following table.

S1	S0	X	D1	D0	Z
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

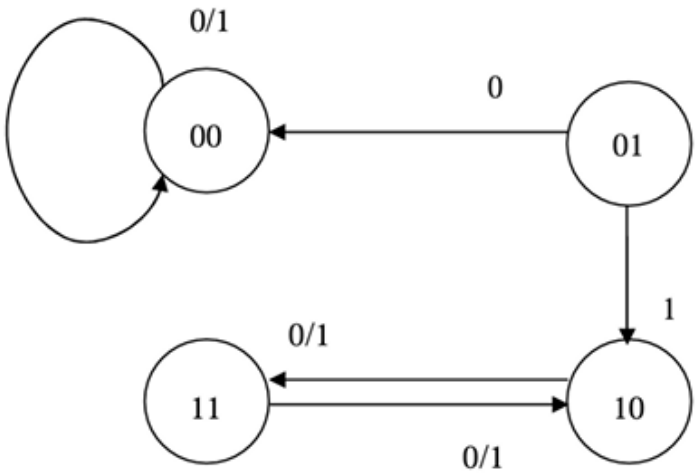
2. Draw the state diagram for the truth table of part 1.

> 数电

1.

S1	S0	X		D1	D0	Z
0	0	0		0	0	0
0	0	1		0	0	0
0	1	0		0	0	1
0	1	1		1	0	1
1	0	0		1	1	1
1	0	1		1	1	1
1	1	0		1	0	1
1	1	1		1	0	1

2.



T7

Suppose a 32-bit instruction takes the following format:

OPCODE	SR	DR	IMM

If there are 56 opcodes and 40 registers, what is the range of values that can be represented by the immediate (IMM)? Assume IMM is a 2's complement value.

假设一个32位指令采取如图所示的格式，有56个操作码，40个寄存器，立即数能代表的范围是多少

56个操作码需要 6 个 bit 表示，40个寄存器一样需要 6 个 bit 表示，则留给立即数的位数：

$$32 - 6 - 6 \times 2 = 14$$

则表示的范围:

$$-2^{13} < IMM < 2^{13} - 1$$

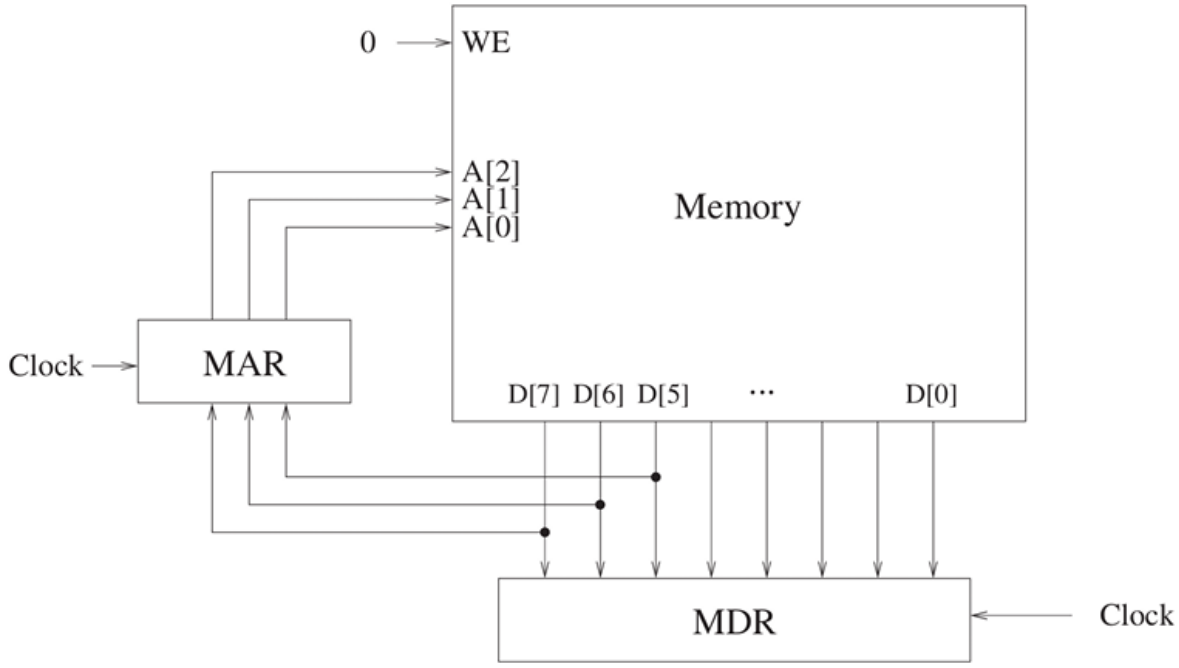
T8

Say it takes 50 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 both the IA-32 instruction "ADD [eax], edx" (refer to) and the LC-3 instruction "ADD R6, R2, R6." Assume each phase (if required) takes one cycle, unless a memory access is required.

> 读写内存需要50个周期，读写寄存器需要1个周期，其它phase 需要一个周期，注意 [] 表示取内存地址的内容
>
> | | 指令 | 取指 | 译码 | 计算地址 | 取操作数 | 执行 | 访存写回 | 总计 |
> | --- | --- | --- | --- | --- | --- | --- | --- | --- |
> | x86 | ADD [eax] edx | 50 | 1 | 1 | 50 | 1 | 50 | 153 |
> | LC-3 | ADD R6, R2, R6 | 50 | 1 | - | 1 | 1 | 1 | 54 |
>
> 做这种类型的题目要注意题目描述，题目中有说明 一个 phase 需要一个周期，如果一个 phase 总包含读写内存则需要50周期

T9

Shown below is a byte-addressible memory consisting of eight locations, and its associated MAR and MDR. Both MAR and MDR consist of flip-flops that are latched at the start of each clock cycle based on the values on their corresponding input lines. A memory read is initiated every cycle, and the data is available by the end of that cycle.



Just before the start of cycle 1, MAR contains 000, MDR contains 00010101, and the contents of each memory location is as shown.

Memory Location	Value
x0	01010000
x1	11110001
x2	10000011
x3	00010101
x4	11000110
x5	10101011
x6	00111001
x7	01100010

1. What do MAR and MDR contain just before the end of cycle 1?
2. What does MDR contain just before the end of cycle 4?

(1) 在第一个周期即将结束前，MAR和MDR的内容

第一个周期 从MAR取出 000 访问 x0 处的内容，此时 MDR 变成01010000，MAR是MDR的前三位即 x2

(2) 在第四周期即将结束前，MDR的内容

第一个周期 MDR 01010000 MAR x2

第二个周期 MDR 10000011 MAR x4

第三个周期 MDR 11000110 MAR x6

> 第四个周期 MDR 00110001 MAR x1
>
> 根据上面的分析 MDR 00110001

作业3

T1

Your task is to consider the successor to the LC-3. We will add 16 addition opcodes to the ISA and the size of register set remains 8. The memory is byte-addressable, with total address space of 32K bytes. Instructions will remain 16 bits wide, though we may need to change the size of some of the fields.

1. How many bits do we need in the PC to be able to address all of memory?
2. What is the largest immediate value we could represent in an ADD instruction on this new machine?
3. What is the address range that a LD instruction is able to cover on this new machine?

> 构思新款的LC-3，我们添加15个操作码，寄存器的个数仍然是8，内存是字节可寻址的，总地址空间为32K字节，指令长度仍然是16位
>
> (1) PC的位数应该变成多少：内存字节可寻址，总地址空间32K，则PC位数：$32K = 2^{15}$
>
> (2) 在这台新机器上，ADD指令中立即数表示的最大值：
>
> 操作码 32 个 需要5位；寄存器 8 个 需要3位；留给立即数 16 - 5 - 3 - 3 -1 = 4位，则最大为 0111 为 7
>
> (3) 同上题的分析，留给LD指令立即数的位数：16 - 5 - 3 = 8 位，则范围为
>
> $$[PC - 128, PC + 127]$$

T2

参照ans3中的解析，已经写的很详细了，注意各个指令是否存在哪些周期是存在争议的，现在按照本次作业中的解析为准，注意第二版和第三版的课本也是存在区别的，为方便大家查看，所以将解析粘贴在这里

> LC-3指令周期包括六个阶段：取指令，译码，计算地址，取操作数，执行，存储结果。 但不是所有指令都需要所有的六个阶段。
>
> 1. 简述指令周期中的各个阶段，以及各阶段分别执行什么操作。
>   1. 取指令：从内存中获取下一条指令，载入指令寄存器。
>   2. 译码：根据指令内容决定哪部分的微结构需要工作
>   3. 计算地址：计算指令执行所需的**内存**地址
>   4. 取操作数：获取指令执行所需的源操作数
>   5. 执行：进行指令的执行
>   6. 存储结果：将执行结果写入目的地
> 2. 填写下表。若某种指令执行时必须经过某阶段，则在对应位置打勾。

	FETCH	DECODE	EVALUATE ADDRESS	FETCH OPERANDS	EXECUTE	STORE RESULT
ADD	✓	✓		✓	✓	
AND	✓	✓		✓	✓	
ST	✓	✓	✓	✓		✓
STR	✓	✓	✓	✓		✓
LDI	✓	✓	✓	✓		✓
LEA	✓	✓			✓	
BR	✓	✓	✓		✓	
JMP	✓	✓	✓		✓	

解析： 首先，所有指令都需要取指和译码。

- ADD：
    1. 不需要evaluate address：课本P131，the ADD and AND instructions in the LC-3 ..... For those instructions, the EVALUATE ADDRESS phase is not needed.
    2. 需要fetch operands：课本P132，In the ADD example, this phase consisted of obtaining the source operands from R2 and R6.
    3. 需要execute：课本P132，In the ADD example, this phase consisted of the step of performing the addition in the ALU.
    4. ★ 不需要store result ：课本P132，the LC-3 ADD instruction does not require a separate EVALUATE ADDRESS phase or a separate STORE RESULT phase.

- AND：和ADD相同
- ST：和LD相同
- STR：与ST相同
- LDI：与LD相同
    1. 需要evaluate address：课本P131，This calculation was performed during the EVALUATE ADDRESS phase.
    2. 需要fetch operands：课本P131，In the LD example, this phase took two steps: loading MAR with the address calculated in the EVALUATE ADDRESS phase and reading memory that resulted in the source operand being placed in MDR.
    3. ★ 不需要execute ：课本P132，The LC-3 LD instruction does not require an EXECUTE phase.
    4. 需要store result
- LEA：（这条指令的执行过程，课本中讲得不太清楚，我也不太确定🍃以下是我自己的理解）
    1. 不需要evaluate address：不需要访问内存，所以不需要计算地址
    2. 不需要fetch operands ❓：两个操作数分别是立即数和PC，不需要额外从寄存器或者内存中读取操作数
    3. 需要execute：需要将两个操作数相加
    4. 不需要store result ❓：类似ADD
- BR：
    1. 需要evaluate address：课本P163，then the PC is loaded with the address obtained in the EVALUATE ADDRESS phase.
    2. 不需要fetch operands：没有操作数需要读取。条件码是在 execute 阶段检查的。
    3. ★需要execute：课本P162，the conditional branch's execute phase either does nothing or it loads the PC with the address of the instruction it wishes to execute next.
    4. 不需要store results：没有结果需要存储
- JMP：类似BR，除了execute阶段不需要检查条件码，直接向PC装载目标地址即可。

3. 假设取指令需要3个周期，译码需要1个周期，执行需要1个周期。 如果需要读写内存，则获取操作数或存储需要100个周期；否则，读写寄存器只需要1个周期。 关于求值地址，PC相对模式和基地址+偏移模式需要1个周期， 而间接寻址模式则需要100个周期。 你的任务是计算以下LC-3程序运行所需的周期数。

Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x30F6	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	1
x30F7	0	0	0	1	0	1	0	0	0	1	1	0	1	1	1	0
x30F8	0	0	1	1	0	1	0	1	1	1	1	1	1	0	1	1
x30F9	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0
x30FA	0	0	0	1	0	1	0	0	1	0	1	0	0	1	0	1
x30FB	0	1	1	1	0	1	0	0	0	1	0	0	1	1	1	0
x30FC	1	0	1	0	0	1	1	1	1	1	1	1	0	1	1	1

答案与解析： 441个周期。

1. LEA，6个周期
2. ADD，6个周期
3. ST，106个周期
4. AND，6个周期
5. ADD，6个周期
6. STR，106个周期
7. LDI，205个周期

4. 考虑ADD, STR和BR三种指令，完成下表。 如果对应的组件在相应的阶段可能被读写，则填入“R”或/和“W”。

	FETCH INSTRUCTION	DECODE	EVALUATE ADDRESS	FETCH OPERANDS	EXECUTE	STORE RESULT
PC	R, W		R `BR`		W `BR`	
IR	W	R	R `STR` `BR`	R `ADD`		
MAR	R, W		W `STR`			R `STR`
MDR	W, R			W `STR`		R `STR`
MEM	R					W `STR`
registers			R `STR`	R `ADD` `STR`	W `ADD`	
NZP					R `BR`, W `ADD`	

5. 995.

解析： 题中描述的是流水线CPU。

T3

Suppose the following LC-3 program is loaded into memory starting at location x3000:

```
x3000: 0101 000 001 1 00000
x3001: 0001 000 000 1 00001
x3002: 0001 010 001 0 00 000
x3003: 0001 010 001 0 00 010
x3004: 1001 010 010 111111
x3005: 0000 010 000000001
x3006: 1111 0000 00100101
x3007: ......
```

1. Suppose the code at x3007 is executed. What is the initial value of R1?
2. We would like to add XOR operation with the unused opcode 1101. The format of XOR instruction is the same as ADD and AND. Rewrite the above program using the XOR instruction at least once.
3. What changes should we make to the current architecture to implement XOR instruction?

假设上述的LC-3程序从地址x3000开始

```
AND R0, R0, #0
ADD R0, R0, #1
; R0 此时值为 1
ADD R2, R1, R0
ADD R2, R1, R2
; R2 = R1 + R1 + 1
NOT R2, R2
; 若 R2 的值为 0xffff 则跳转
BRz #1
HALT
```

(1) 加入x3007处的指令被执行，R1的初值是多少： R1初值可是 0xffff 和 0x7fff

(2) 添加XOR指令，操作码 1101，重写程序：将 `ADD R2, R1, R2` 更改为 `XOR R2, R1, R2`

(3) 增加XOR指令需要对架构做出什么改动：ALU添加异或操作；控制模块添加异或控制信号

T4

In this problem we perform five successive accesses to memory. The following table shows for each access whether it is a read (load) or write (store), and the contents of the MAR and MDR at the completion of the access. Some entries are not shown. Note that we have shortened the addressability to 5 bits, rather than the 16 bits that we are used to in the LC-3, in order to decrease the excess writing you would have to do.

Operations on Memory

	R/W	MAR	MDR				
Operation 1	W		1	1	1	1	0
Operation 2							
Operation 3	W		1	0			
Operation 4							
Operation 5							

The following five tables show the contents of memory locations x4000 to x4004 before the first access, after the 3nd access and after the 5th access. Again, not all entries are shown. We have added an unusual 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.

Memory before Access 1

x4000	0	1	1	0	1
x4001	1	1	0	1	0
x4002		1			
x4003	1	0	1	1	0
x4004	1	1	1	1	0

Memory after Access 3

x4000					0
x4001		0			0
x4002					
x4003					
x4004	1	1	1	1	0

Memory after Access 5

x4000					
x4001					
x4002					
x4003	0	1	1	0	1
x4004	1	1	1	1	0

根据信息猜测 填表，核心是 MDR 只会在 load 指令后更改

- 在 op3 之后 MDR的值改变了，那么 op2 的操作一定是 R
- 在 op3 之后 x4001的值改变了 且第二位和第五位是0 则 op3 只能是向 x4001 写入值
- 根据写入的值0的位置，判断op2只能是读取x4003的值 为 10110 ， op3 的 MDR 也是该值，也可以填上 在op3操作之后 x4001的值
- 因为op2还需要从x4003读值，则op1写入的地址一定不是x4003，所以可以填写op3操作后x4003的值
- 根据op5后的信息，x4003发生改变了，则一定需要先读写改变MDR的值，再写入x4003，根据此可以填出op4和op5的操作以及MDR值，以及op5操作的地址
- 反推op4需要读取到01101，根据op3之后的内存信息，只能从x4002读取，可以填写op3后x4002的值为01101，同样可以填写op5之后x4001和x4002的值

- 最后由于op3之后x4000的值发生了改变，显然op1操作写入的地址是x4000，x4002的初值是01101

T5

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 ".

```
1   AND R0, R0, #0
2   ADD R0, R0, #2
3   BRzp X
4   ADD R0, R0, #3
5   ADD R0, R0, #1
6   HALT
```

这道题可以程序验证的 解析略 注意是需要运行到 HALT 时候 R0 的值

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

作业4

T1

Many ISAs have a conditional load instruction (LDC), which loads a value from memory into a register based on the condition codes. We could add that instruction to the LC-3 ISA using the unused opcode. Further we could use the BEN bit (BEN = (IR[11] AND N) OR (IR[10] AND Z) OR (IR[9] AND P)) the same way we use BEN to determine whether to take the conditional branch. The LDC instruction has three operands: DR, PC offset, and the nzp bits.

If a program contained an LDC instruction in memory location x4000, what is the largest memory address that can provide the value to be loaded into DR?

> 前面那么多都没啥用 关键是指令格式 DR PC offset nzp，留给offset的位数 16-4-3-3 =6
>
> 所以：
>
> $$x4001 + (011111)_2 = x4020$$

T2

List five addressing modes in LC3. Given instructions ADD, NOT, LEA,LDR and JMP, categorize them into operate instructions, data movement instructions, or control instructions. For each instruction mentioned above, list addressing modes that can be used

> Addressing modes: register, immediate, PC-relative, indirect, Base+offset
>
Instruction	Type	Addr mode(s)
ADD	operate	register, immediate
NOT	operate	register
LEA	data movement	immediate
LDR	data movement	Base+offset
JMP	control	register

T3

In class we showed the first few states of the finite state machine that is required for processing instructions of a computer program written for LC-3. In the first state, the computer does two things, represented as:

```
MAR <-- PC
PC <-- PC+1
```

1. Why does the microarchictecture put the contents of the PC into the MAR?
2. Why does the microarchitecture increment the PC?

> (1) 计算机会根据MAR的值去内存中取指令(差不多意思就行)
>
> (2) 指向下一条需要被执行的指令的地址(差不多意思就行)

T4

1. Write a **single** LC3 assembly instruction that copies the content of `R5` to `R4`.
2. Write a **single** LC3 assembly instruction that clears the content of `R3` (i.e. `R3 = 0`).
3. Write **3** LC3 assembly instructions that does `R1 = R6 - R7`.
   - You are ONLY allowed to change the value of `R1`.
   - You may assume that the initial value of `R1` is 0.
4. Write **3** LC3 assembly instructions that multiply the value at label `DATA` by 2. (`Mem[DATA] = Mem[DATA] * 2`)
   - You are ONLY allowed to change the value of `R1`.
   - You don't need to restore or clear the value of the register you used.
   - No need to consider overflow.
5. Set condition codes based on the value of `R1` using only `one` LC-3 instruction.
   - You are not allowed to change any value in the registers.

```
;1
ADD R4, R5, #0 ;或者
AND R4, R5, #-1
;2
AND R3, R3, #0
;3
NOT R1, R7
ADD R1, R1, #1
ADD R1, R1, R6
;4
LD R1, DATA
ADD R1, R1, R1
ST R1, DATA
;5
ADD R1, R1, #0 ;或者
AND R1, R1, #-1
```

T5

How many times does the LC-3 make a read or write request to memory during the processing of the LD instruction?

How many times during the processing of the LDI instruction? How many times during the processing of the LEA instruction?

Also indicate what phases the instructions don't need. Processing includes all phases of the instruction cycle.

> `LD`：2 次，取指令时，取数据时
>
> `LDI`：3 次，取指令时，取数据存储的地址，取数据时
>
> `LEA`：1 次，取指令时

T6

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?

> (1) ADD和AND指令会获得更大的立即数范围，但是对NOT指令不产生影响
>
> (2) LD和ST指令的寻址范围更大
>
> (3) BR指令中不涉及寄存器，所以不会有变化

T7

Write a short LC-3 program that compares the two numbers in `R1` and `R2` and puts the value 0 in `R0` if `R1 = R2`, 1 if `R1 > R2`, and −1 if `R1 < R2`.

```
1           .ORIG x3000
2           NOT R3, R2
3           ADD R3, R3, #1
4           ADD R3, R3, R1
5           BRz EQUAL
6           BRp GREATER
7   LESS    LD R0, NEG_ONE
8           BR END
9   EQUAL   LD RO, ZERP
10          BR END
```

```
11    GREATER LD R0, ONE
12    END     HALT
13    ZERO    .FILL #0
14    ONE     .FILL #1
15    NEG_ONE .FILL #-1
16            .END
```

T8

The content in PC is x3010. The content of the following memory unit is as follows

Address	Value
x304E	x70A4
x304F	x70A3
x3050	x70A2
x70A2	x70A4
x70A3	x70A3
x70A4	x70A2
x3010	1110 0110 0011 1110
x3011	0110 1000 1100 0001
x3012	0110 1111 0000 0001
x3013	0110 1101 1111 1111

1. After the execution of the following code, What is the value stored in `R6` ?
2. Can you use one LEA instruction to do the same task as the three instructions above do?(Only consider loading value into R6.)

(1)

- LEA 指令，偏移量为 x3E，计算得到地址 0x304F 放入 R3
- LDR 指令，偏移量为 x01，从 R3 中读取地址计算，得到地址 0x3050，读取内存将0x70A2 放入 R4
- LDR 指令，偏移量为 x01，从 R4 中读取地址计算，得到地址 0x70A3，读取内存将0x70A3 放入 R7
- LDR 指令，偏移量为 11 1111，从R7中读取地址计算，得到地址 0x70A2，读取内存将0x70A4 放 入 R6

(2) 不可以，LEA的偏移量位数不足

T9

Shown below are the contents of registers before and after the LC-3 instruction at location x3210 is executed. Your job: Identify the instruction stored in x3210. Note: There is enough information below to uniquely specify the instruction at x3210

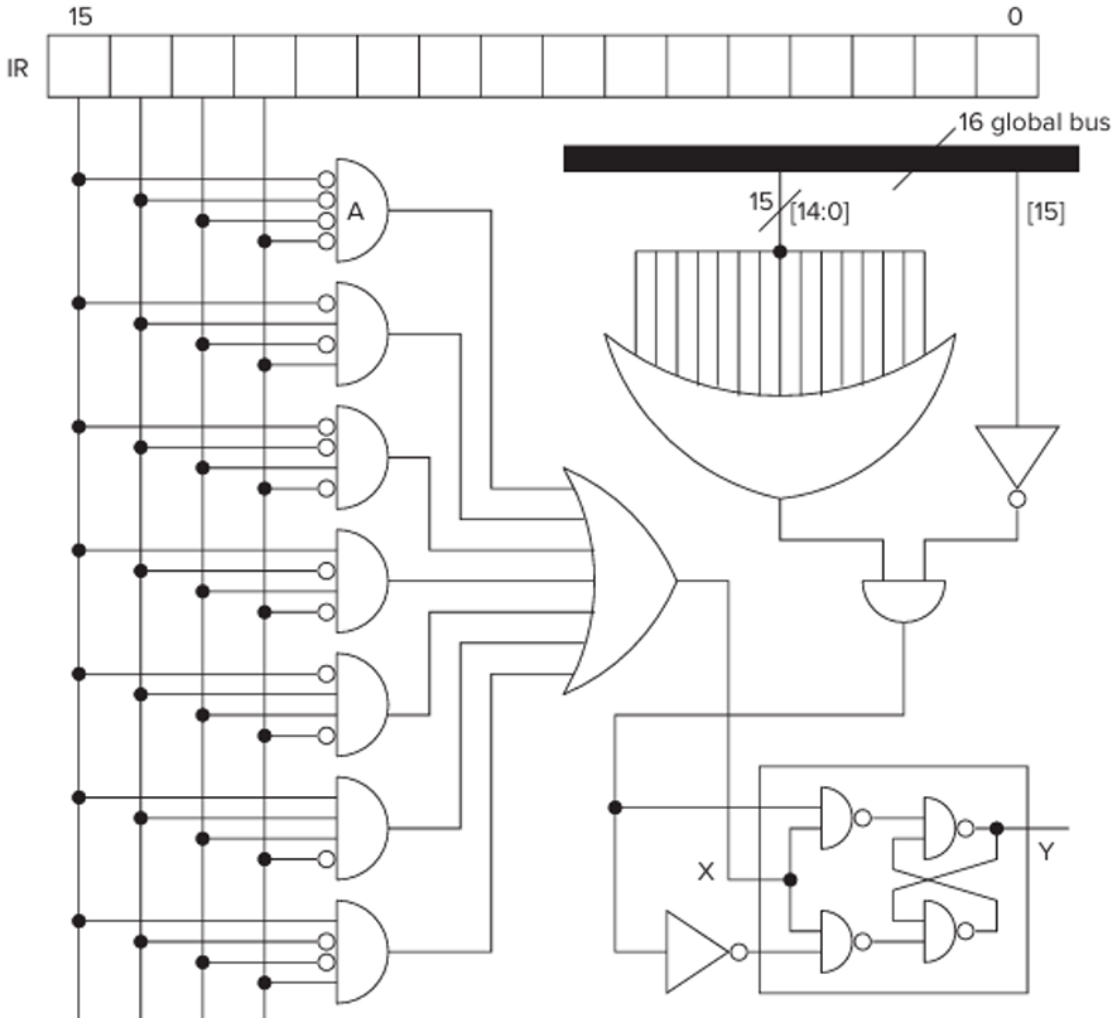
	Before	After
R0:	xFF1D	xFF1D
R1:	x301C	x301C
R2:	x2F11	x2F11
R3:	x5321	x5321
R4:	x331F	x331F
R5:	x1F22	x1F22
R6:	x01FF	x01FF
R7:	x341F	x3211
PC:	x3210	x3220
N:	0	0
Z:	1	1
P:	0	0

> 更改了R7的值，且PC没有正常+1，可以想到LC-3发生了跳转，又因为R7的值刚好等于PC+1，则可以判断执行的指令是： JSR #15 （JSR指令会跳转到PC+offset的位置，并将跳转前的PC存入R7）

T10

Apart of the implementation of the LC-3 architecture is shown in the following diagram.

1. What information does Y provide?
2. The signal X is the control signal that gates the gated D latch. Is there an error in the logic that produces X?



> 先观察锁存器：10置0；01置1；11保持
>
> 再根据从bus获取的符号位，符号位是1时输出10置0，符号位是0时，且其他位不全为0时，输出01置1，X输入在指令操作码不为0000，0101，0010，1010，0110，1110，1001指令时，输出11，Y的输出会保持，说明 Y 的输出代表 P 状态码
>
> 0000和1110操作码并不会更新状态码，而且0001会更新状态码，需要更改逻辑

作业5

T1

What is the purpose of the `.END` pseudo-op? How does it differ from the `HALT` instruction?

> .END是程序结束的标识
>
> .END 由汇编器处理，之后的指令会被忽视，但是HALT后的指令仍然会被汇编器处理 (言之有理)

T2

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`, for NO OPERATION. The instruction is fetched, decoded, and executed. The execution phase is to do nothing! Which of the following three instructions could be used for `NOP` and have the program still work correctly?

a. 0001 001 001 1 00000

b. 0000 111 000000001

c. 0000 000 000000000

> a. ADD R1, R1, #0 会设置条件码
>
> b. BRnzp #1 无条件向前跳转一个PC 注意 这会忽视一个指令 因为PC已经+1
>
> c. 无事发生，等同于 NOP

T3

Suppose you write two separate assembly language modules that you expect to be combined by the linker. Each module uses the label `AGAIN`, and neither module contains the pseudo-op `.EXTERNAL AGAIN`.

Is there a problem using the label `AGAIN` in both modules? Why or why not?

> 没有影响，每一个模块拥有自己的 symbol table，能够标识不同的地址

T4

Two students wrote interrupt service routines for an assignment. Both service routines did exactly the same work, but the first student accidentally used `RET` at the end of his routine, while the second student correctly used `RTI`.

There are three errors that arose in the first student's program due to his mistake. Describe all of them.

> RET 是无条件跳转到 R7 寄存器存储的地址 RTI 则会在弹出栈中的 PC 和 PSR 后跳转
>
> - 栈空间不平衡
> - 特权模式和条件码没有恢复（存储在 PSR 程序状态寄存器中）
> - 由于根据了 R7 寄存器的内容跳转，而不是栈中的值，跳转会不正常

T5

The input stream of a stack is a list of all the elements we pushed onto the stack, in the order that we pushed them. The output stream is a list of all the elements that are popped off the stack in the order that they are popped off.

(1) If the input stream is ABCD, create a sequence of pushes and pops such that the output stream is BDCA.

(2) If the input stream is ABCD, is it posible to create a sequence of pushes and pops such that the output stream is DBCA? Why?

(3) If the input stream is ABCDEF, how many different output streams can be created? Only consider output streams that are 6 characters long.

(1) 输入是ABCD 设计一个输出是BDCA的栈操作过程：PUSH PUSH POP PUSH PUSH POP POP POP

(2) 输入是ABCD 设计一个输出是DBCA的栈操作过程：不可能 D第一个出栈说明全部元素已经入栈，不可能先弹出B再弹出C

(3) 输入是ABCDEF，有多少种输出序列：132 ~~*感觉这道题出在本门课程中不太合适就是了*~~

考虑一个有 n 个元素的输入序列，设它们为：

$$a_1, a_2, a_3, \ldots, a_n$$

假设元素 a_1 是第 j 个出栈的元素，那么根据栈的性质，在 a_1 出栈之前，比 a_1 先进栈的元素肯定已经全部出栈了，它们是：

$$a_2, a_3, \ldots, a_j$$

而在 a_1 出栈之后，栈一定是空的，尚未入栈的元素为：

$$a_{j+1}, a_{j+2}, \ldots, a_n$$

设 $f(n)$ 表示长度为 n 的序列的输入的出栈序列种类，那么元素 $a_2 \sim a_j$的出栈序列种类为 $f(j-1)$，同样对于剩余的元素 $a_{j+1} \sim a_n$，出栈序列种类为 $f(n-j)$，根据乘法原理有：a_1在第 j 个位置出栈的出栈序列：

$$f(j-1) \times f(n-j)$$

那么可以根据此求出 $f(n)$：

$$f(n) = \sum_{j=0}^{n} f(j-1) \times f(n-j)$$

至于这个递推式的通项公式，助教也不会求，感兴趣的同学可以去搜索卡特兰数的证明

$$f(n) = \frac{1}{1+n} C_{2n}^{n}$$

T6

(1) When the following LC-3 program is executed, how many times will the instruction at the memory address labeled `LOOP` execute?

```
        .ORIG x3005
        LEA R2, DATA
        LDR R4, R2, #0
LOOP    ADD R4, R4, #-3
        BRzp LOOP
        TRAP x25
DATA    .FILL x8002
        .END
```

(2) Now if we replace the instruction in x300A with `DATA .FILL x8003`, will your answer be the same?

> (1) x8002=32770, 32770/3 = 10923.3 因此 10924次
>
> (2) 1次 因为第一次计算后，R4的值变为 x8000 就直接跳出循环了

T7

Figure 8.18 in P286 describes a FIB subroutine as shown below:

```
;FIB subroutine
; + FIB(0) = 0
; + FIB(1) = 1
; + FIB(n) = FIB(n-1) + FIB(n-1)
;
; Input is in R0
; Return answer in R1
;
FIB     ADD R6, R6, #-1
        STR R7, R6, #0 ; Push R7, the return linkage
        ADD R6, R6, #-1
        STR R0, R6, #0 ; Push R0, the value of n
        ADD R6, R6, #-1
        STR R2, R6, #0 ; Push R2, which is needed in the subroutine
; Check for base case
        AND R2, R0, #-2
        BRnp SKIP ; Z=0 if R0=0,1
        ADD R1, R0, #0 ; R0 is the answer
        BRnzp DONE
; Not a base case, do the recursion
SKIP    ADD R0, R0, #-1
        JSR FIB ; R1 = FIB(n-1)
        ADD R2, R1, #0 ; Move result before calling FIB again
        ADD R0, R0, #-1
        JSR FIB ; R1 = FIB(n-2)
        ADD R1, R2, R1 ; R1 = FIB(n-1) + FIB(n-2)
; Restore registers and return
DONE    LDR R2, R6, #0
        ADD R6, R6, #1
        LDR R0, R6, #0
        ADD R6, R6, #1
        LDR R7, R6, #0
        ADD R6, R6, #1
        RET
```

(1) Is R0 caller save or callee save? What about R2 and R7?

(2) The following table shows the instruction cycles used to execute this subroutine for the input n:

n	2	3	4	5	6	7	8	9	10
$T(n)$	55	93	169	283	473	777	1271	2069	3361

Can you define $T(n)$ recursively assuming n is sufficiently large, and explain your definition?

(3) How can you improve the efficiency of this *recursive subroutine*? Do not use iterative methods instead.

(1) 均为 callee

- callee save为被调用者保存的数据，caller为调用者保存的数据，在本道题中R0 R2 R7都是在调用后写到栈中的，所以均为 callee
- 过去我错误的认为传的参数由调用者保存，返回地址由被调用者保存，在LC-3中不是这样的

(2) 时钟周期一定是以下的格式，待定系数 $C = 21$

$$T(n) = T(n-2) + T(n-1) + C$$

(3) 如何提升效率：将 $FIB(n)$ 的值保存在内存中，在需要用到时先去内存中找，否则递归计算

T8

We have a program with some missing instructions, and we have a table consisting of some information and some missing information associated with five specific clock cycles of the program's execution. Your job is to complete both!

(1) Insert the missing instructions in the program and the missing information in the table. Cycle numbering starts at 1. That is, cycle 1 is the first clock cycle of the processing of `LD R0, A`. Note that we are asking for the value of the registers DURING each clock cycle.

We have not said anything about the number of clock cycles a memory access takes, but you do have enough information to figure that out for yourself.

```
        .ORIG x3000
        LD R0, A
        LD R1, B
        NOT R1, R1
        ADD R1, R1, #1
        AND R2, R2, #0
AGAIN   -------------- (a)
        -------------- (b)
        BRnzp AGAIN
DONE    ST R2, C
        HALT
A       .FILL #5
B       .FILL -------- (c)
C       .BLKW #1
        .END
```

Cycle Number	State Number	Information
(d)	(e)	LD.REG: 1 DRMUX: (f) GateMDR: (g) LD.CC: (h) GateALU: (i) GatePC: (j)
16	30	LD.MDR: (k) MDR: (m) IR: (n) LD.IR: (l)
50	(o)	LD.REG: 1 MDR: (p) x_4A_ DRMUX: (q) BUS: x0001 GateMDR: (r)
57	1	PC: (s) IR: (t) x_040 GateALU: (u) BUS: x0003 GatePC: (v)
(w)	22	ADDR1MUX: (x) ADDR2MUX: (y) LD.PC: 1 PC: x3008 PCMUX: ADDER

(2) Actually, the program was written by a student, so as expected, he did not get it quite right. Almost, but not quite! Your final task on this problem is to examine the code, figure out what the student was trying to do, and point out where he messed up and how you would fix it. It is not necessary to write any code, just explain briefly how you would fix it.

- 观察表格第四行，状态 1 是 ADD 指令，则可以分析出 t 的值为 x1040，则一定有个指令是 ADD R0, R1, R0，而给出的程序代码中没有这一指令，则可以确定 a/b 其中一个的指令
- 观察第三行的 LD.REG 说明此时需要对寄存器赋值，而与第四行的周期只差了7，说明上一条指令一定是简单的 ADD AND NOT (执行需要5+x周期，x为访存所需周期)，观察当前的MDR寄存器，x_4A_，说明指令的格式应当是对R2执行操作（且一定是立即数）并存回R2。给出的指令 AND R2, R2, #0似乎符合要求，但是观察当前bus的值为x0001，而AND R2, R2, #0指令的结果只能是0，说明当前指令是隐藏的另外一个指令。即R2在和立即数进行一个操作后变成了1，又因为0 and任何数都是0，那么该隐藏指令只能是ADD，该指令为 ADD R2, R2, #1，那么可以确定a和b的值，此时可以填写一堆值。
- 对于第三行数据，首先 p 的值可以确认是 x14A1，DRMUX表示目标寄存器，显然此时需要写入的寄存器是R2，那么 q 的值是010，GATEMDR表示是否需要从内存中加载数据，ADD指令是不需要的，所以 r 的值为 0，同时根据 LD.REG=1可以判断此时处于状态 1，o 的值为 1；

- 对于第四行数据，GateALU表示ALU计算的数据是否需要传给总线，此时显然是需要的，u 的值为 1，GatePC的意思是 PC 的值是否需要传输给总线，此时是不需要的，v 的值为 0。
- 那么可以分析第三行第四行数据两个状态 1 之间差了 7 个周期，说明一个 ADD 指令需要 7 个周期，即对内存的访问需要 2 个周期完成。
- 知道了这些我们可以开始回推，LD指令现在需要11个周期。第一行中 LD.REG = 1，说明此时在状态 27，那可以填写 d 的值为 11，e 的值为 27；LD指令需要设置状态码，操作的寄存器是R0，不需要从ALU输出值，需要从MDR输出值，不需要从PC输出值，则可以依次填出 hfigj 分别为 1 000 0 1 0
- 表格第二行对应的第二条 LD 指令，状态是 MDR 的值赋给 IR，那么此时 MDR 的值应该是第二条 LD 指令，即 x2209，而 IR 的值应当是上一条指令，即 x2009，该状态不需要对 MDR 执行赋值，需要对 IR 进行赋值，所以 LD.MDR 为 0，LD.IR 为 1
- 最后一行状态 22 发生在 BR 指令中，则当前的时钟周期为 65 ，那么 PC 需要从 PCMUX 更新值，则 LD.PC 的值应该为1，而在BR指令中，加法的两个操作数是 PC 和 九位的 offset，那么 ADDR1MUX 的值为 PC，ADDR2MUX 的值为 PCoffset9
- 最后观察整个程序是计算A/B，并将结果存储在C中，根据第四行总线上的值，可以判断内存地址B处存的值为 2

第二问，显然 BRnzp 指令永远不会结束循环，更改为 BRp

附录

IEEE浮点数

- 半精：1 位符号位 5 位阶码 10位尾数
- 32位：1 位符号位 8 位阶码 23位尾数
- 64位：1 位符号位 11 位阶码 52位尾数

阶码中的那个减数怎么取：能表示的最大正数，比如5位就是0 1111，8位就是0111 1111

关于各种phase的判断

- 以第三次作业中的题目为准，如果没有具体标出的部分，根据自己的理解来，没有一个绝对的正确答案

特权模式和用户模式

- 上课中没有说清楚的情况：
  - 如何进入特权模式：通过中断/系统调用 没有直接在代码中直接调用的方式
  - 关于调用栈，R6会在用户模式和特权模式自动指向用户的栈空间和系统的栈空间