Homework：存储

> 本此作业统一以 $K=10^3, M=10^6, G=10^9$ 为计量单位。以后如果遇到类似的题目请提前查看/询问/确定这件事。

> 请直接用 Markdown 题目源文件填充答案，最后统一提交 PDF 格式，比如使用 Typora 导出。

T1

以下是A型号磁盘的相关参数

参数	值
盘片数	2
每个盘片的面数	2
柱面数（也叫磁道数）	500000
平均每条磁道的扇区数	500
扇区大小	4096 Byte
旋转速率	5400 RPM
平均寻道时间	10ms

1.1

求该磁盘的容量（GB 为单位）。

答案:

磁盘容量 = 字节数/扇区 * 平均扇区/磁道 * 磁道/表面 * 表面/盘片 * 盘片/磁盘

$$\begin{aligned} Capacity &= 4096 \times 500 \times 500000 \times 2 \times 2 \\ &= 4096GB \\ &= 4.096TB \end{aligned}$$

典型错误

$$\begin{aligned} \text{磁道容量} &= \frac{\text{盘片数}}{\text{磁盘}} \times \frac{\text{表面数}}{\text{盘片}} \times \frac{\text{磁道数}}{\text{盘面}} \times \frac{\text{扇区数}}{\text{磁道}} \times \frac{\text{字节数}}{\text{扇区}} \\ &= \frac{2\text{盘片}}{\text{磁盘}} \times \frac{2\text{盘面}}{\text{盘片}} \times \frac{500000\text{磁道}}{\text{盘面}} \times \frac{500\text{扇区}}{\text{磁道}} \times \frac{4096\text{字节}}{\text{扇区}} \\ &= 4096 \times 10^9 Byte (\approx 4096GB) \end{aligned}$$

> 前面都好好的，这个约等于错了，这就是等于

求该磁盘的容量（GB 为单位）。
总容量 = 盘片数×每个盘片的面数×柱面数×每条磁道的扇区数×扇区大小 = 4000GB

1.2.1

求该磁盘访问一个扇区的平均延迟（ms 为单位）。

> 4096000000000 就是 4096GB

答案：

$$T_{AvgSeek} = 10ms$$

$$T_{AvgRotation} = 0.5 \times \frac{1}{5400RPM} \times 60000ms = 5.56ms$$

$$T_{AvgTransfer} = \frac{1}{5400RPM} \times \frac{1}{500} \times 60000ms = 0.02ms$$

$$T_{access\_one\_sector} = T_{AvgSeek} + T_{AvgRotation} + T_{AvgTransfer} = 15.58ms$$

> 实际上是 $15.5\dot{7}$ ms

1.2.2

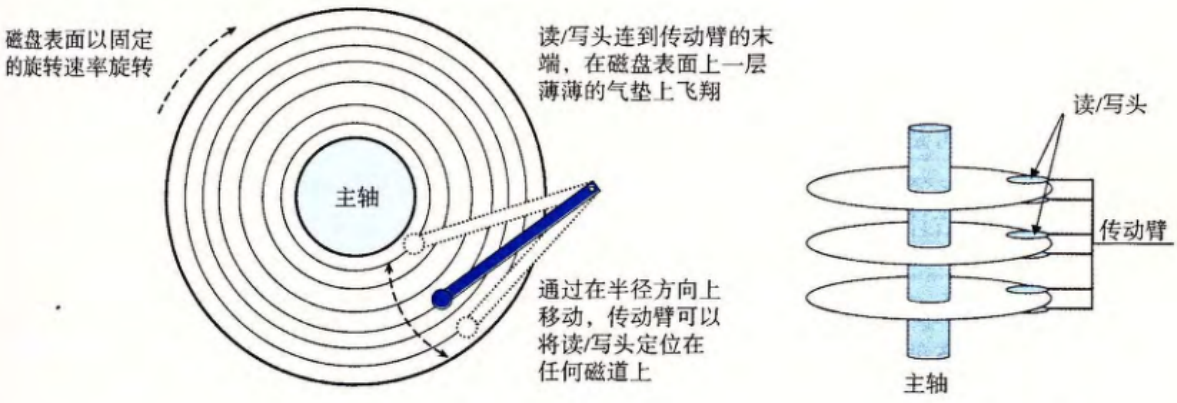
求该磁盘随机读写时的每秒访问次数（IOPS）。

> 提示：思考磁盘的最小访问单位是什么，书上有提到

答案：

$$IOPS = \frac{1}{T_{access\_one\_sector}} \times \frac{1000ms}{s} \times 1s = 64.2$$

> 这是因为磁盘的最小访问单位是一个扇区，每次可做的最小粒度的 I/O 操作是一个扇区的读写。
>
> 因为是随机读写，所以寻道，旋转等待的开销是显然要算上的。
>
> 思考如果我要读的数据是一个扇区的一部分会发生什么：仍然读整个扇区，只是其他数据你不要而已。



a）一个盘片的视图　　b）多个盘片的视图

图 6-10　磁盘的动态特性

在传动臂末端的读/写头在磁盘表面高度大约 0.1 微米处的一层薄薄的气垫上飞翔(就是字面上这个意思)，速度大约为 80 km/h。这可以比喻成将一座摩天大楼(442 米高)放倒，然后让它在距离地面 2.5 cm(1 英寸)的高度上环绕地球飞行，绕地球一天只需要 8 秒钟！在这样小的间隙里，盘面上一粒微小的灰尘都像一块巨石。如果读/写头碰到了这样的一块巨石，读/写头会停下来，撞到盘面——所谓的读/写头冲撞(head crash)。为此，磁盘总是密封包装的。

磁盘以扇区大小的块来读写数据。对扇区的访问时间(access time)有三个主要的部分：寻道时间(seek time)、旋转时间(rotational latency)和传送时间(transfer time)：

- **寻道时间**：为了读取某个目标扇区的内容，传动臂首先将读/写头定位到包含目标扇区的磁道上。移动传动臂所需的时间称为寻道时间。寻道时间 T_{seek} 依赖于读/写头以前的位置和传动臂在盘面上移动的速度。现代驱动器中平均寻道时间 $T_{\text{avg seek}}$ 是通过对几千次对随机扇区的寻道求平均值来测量的，通常为 3～9ms。一次寻道的最大时间 $T_{\text{max seek}}$ 可以高达 20ms。

1.2.3

求该磁盘的顺序读取带宽（MB/s 为单位）。

其中，顺序读取带宽的定义是

$$\max\{\forall_{\text{可以存储在磁盘}A\text{上的文件}F}\frac{F\text{的大小}}{\text{磁盘从随机时刻开始，顺序读取完}F\text{所需的期望时间}}\}$$

为了答案统一，有如下假设：

1. 不能确认为 0 的值，都认为是以平均值为期望的随机数（比如即使是顺序地访问磁道，每次寻道时间也是以平均寻道时间为期望的随机数）
2. 顺序存储的文件在相邻的扇区上是连续，当大小超过一个盘面的一个磁道可以容纳的空间时，你可以自己决定下一个开始存储的位置。显然本题你需要想一想什么存法读取时更快。

可以在答案中附上你答案对应的文件的存储方式（可以画图）

答案：

因为切换磁道的同时，磁盘也在转，所以寻道时间一定不为 0，所以即使是切换到相邻的磁道，按照题目假设也需要等待半圈时间。为了尽量减少切换磁道的等待，我们应该最后再切磁道。

所以文件的存法应该是：

1. 先在一个盘面的一个磁道上存满，假设从 0 号扇区开始，存到第 499 个。
2. 切到另一个盘面（无所谓哪个）的相同磁道上，也从 0 号扇区开始，即读写头是不需要等待转的，无缝衔接的（或者说此时读写头在的扇区）。

3. 直到这个磁道的所有扇区都存满（所有的盘片，所有的盘面），我们再切换到下一个磁道。

读法同存法，注意到从 3 开始就是一个循环了，所以我们假设只存在一个磁道上即可。

步骤	耗时	说明
磁头定位时间	$10ms$	寻道时间
旋转等待	$5.56ms$	
顺序读取所有扇区的时间	$2 \times 2 \times 11.11ms$	两个盘面，各两个盘片，都完整转一圈，中间不需要等待
总时间	$60ms$	

此时文件大小为 $4096 \times 500 \times 2 \times 2 = 8.192MB$

所以顺序读取带宽为 $\frac{8.192MB}{60ms} = 136.53MB/s$

T2

下面的表给出了一些不同的高速缓存的参数。你的任务是填写出表中缺失的字段。其中 m 是物理地址的位数，C 是高速缓存大小（数据字节数），B 是以字节为单位的块大小，E 是相联度，S 是高速缓存组数，t 是标记位数，s 是组索引位数，而 b 是块偏移位数。

m	C	B	E	S	t	s	b
32		**4**	**4**		**24**	**6**	
32	**1024**	**32**	**2**				
32	**2048**			**256**	**21**	**8**	**3**

答案：

解题技巧：

1. 先把 S, s；B, b 这种互相推导的填上

m	C	B	E	S	t	s	b
32		**4**	**4**	64	**24**	**6**	2
32	**1024**	**32**	**2**				5
32	**2048**	8		**256**	**21**	**8**	**3**

2. 检查常见公式：$C = BES$ 有没有四缺一，$t = m - (s + b)$ 有没有四缺一

m	C	B	E	S	t	s	b
32	1024	**4**	**4**	64	**24**	**6**	2
32	**1024**	**32**	**2**	16	23	4	5
32	**2048**	8	1	**256**	**21**	**8**	**3**

3. 不停做这两步，直到填满所有的空

T3

假设我们有一个具有如下属性的系统:

- 内存是字节寻址的。
- 内存访问是对 1 字节字的（比如访问一个 char）。
- 地址宽 12 位。
- 高速缓存是两路组相联的(E=2)，块大小为 4 字节(B=4)，有 4 个组(S=4)。

高速缓存的内容如下，所有的地址、标记和值都以十六进制表示:

组索引	标记	有效位	字节1	字节2	字节3	字节4
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0	54	55	56	57
2	00	1	48	49	4A	4B
	40	0	21	22	23	24
3	FF	1	9A	D0	03	EE
	00	0	A1	A2	A3	A4

对于下面每个内存访问，当他们顺序执行时，指出高速缓存是否命中，如果命中且操作前的数可从已有信息判断，请给出。

操作	地址	命中	值（或未知）
读	0x834		
写	0x836		
读	0xFFF		

答案:

操作	地址	命中	值（或未知）
读	0x834	miss	未知
写	0x836	hit	未知
读	0xFFF	hit	EE

```
0x83 = 10000011
0x40 = 01000000
0xFF = 11111111
```

tag	index	offset	解析
10000011	01	00	标记位是 0，没命中，未知
10000011	01	10	命中，但是之前有效位是 0，所以未知
11111111	11	11	命中，EE

易错点：有效位为 0 时，cache line的内容将没有任何意义，比如以下情况，有效位为 0 且缓存不为 0

1. 设备刚刚启动，且该硬件不保证断电重启后每个位置的值，可能是随机的，系统初始化时只把有效位清零
2. 缓存一致性，在多核 CPU 中，一个核心改完某内存后可能会同步给其他 cache 这个位置无效的信息（有兴趣的可以自查 L1, L2, L3 cache）

争议点：本题没有说明写操作的策略，这里应该是写回+写分配。事实上，根据课本的建议，我们总是默认写回+写分配，但如果考试题目没讲清楚还是问一下比较好。

写的情况就要复杂一些了。假设我们要写一个已经缓存了的字 w(写命中，write hit)。在高速缓存更新了它的 w 的副本之后，怎么更新 w 在层次结构中紧接着低一层中的副本呢？最简单的方法，称为直写(write-through)，就是立即将 w 的高速缓存块写回到紧接着的低一层中。虽然简单，但是直写的缺点是每次写都会引起总线流量。另一种方法，称为写回(write-back)，尽可能地推迟更新，只有当替换算法要驱逐这个更新过的块时，才把它写到紧接着的低一层中。由于局部性，写回能显著地减少总线流量，但是它的缺点是增加了复杂性。高速缓存必须为每个高速缓存行维护一个额外的修改位(dirty bit)，表明这个高速缓存块是否被修改过。

另一个问题是如何处理写不命中。一种方法，称为写分配(write-allocate)，加载相应的低一层中的块到高速缓存中，然后更新这个高速缓存块。写分配试图利用写的空间局部性，但是缺点是每次不命中都会导致一个块从低一层传送到高速缓存。另一种方法，称为非写分配(not-write-allocate)，避开高速缓存，直接把这个字写到低一层中。直写高速缓存通常是非写分配的。写回高速缓存通常是写分配的。

程序的程序员来说，我们建议在心里采用一个使用写回和写分配的高速缓存的模型。这样建议有几个原因。通常，由于较长的传送时间，存储器层次结构中较低层的缓存更可能使用写回，而不是直写。例如，虚拟内存系统(用主存作为存储在磁盘上的块的缓存)只使用写回。但是由于逻辑电路密度的提高，写回的高复杂性也越来越不成为阻碍了，我们在现代系统的所有层次上都能看到写回缓存。所以这种假设符合当前的趋势。假设使用写回写分配方法的另一个原因是，它与处理读的方式相对称，因为写回写分配试图利用局部性。因此，我们可以在高层次上开发我们的程序，展示良好的空间和时间局部性，而不是试图为某一个存储器系统进行优化。

T4

仔细阅读下面的程序，根据条件回答下列各题：

- 地址宽度为 10
- 数组的起始地址为 0b0001000000（即二进制表示）
- Block size = 4 Byte，Set = 4，两路组相连（B = 4, S = 4, E = 2）
- 替换算法为 LRU (最近最少使用)

```
#define LENGTH 8
void clear44(char array[LENGTH][LENGTH]) {
    int i, j;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            array[i][j] = 0;
}
```

4.1.1

以上程序会发生几次缓存miss?

答案： 4

4.1.2

如果 LENGTH = 16，那么会发生几次缓存miss?

答案： 4

4.1.3

如果 LENGTH = 17，那么会发生几次缓存miss?

答案： 7

```
S 40,1 miss
S 41,1 hit
S 42,1 hit
S 43,1 hit
S 51,1 miss
S 52,1 hit
S 53,1 hit
S 54,1 miss
S 62,1 miss eviction
S 63,1 hit
S 64,1 miss
S 65,1 hit
S 73,1 miss eviction
S 74,1 miss eviction
S 75,1 hit
S 76,1 hit
hits:9 misses:7 evictions:3
```

4.1.4

请画出在 LENGTH = 17 时，程序执行结束时 set0 和 set1 的高速缓存状态，假设一开始全空。

> 可以用 `Array[0][0] ~ Array[0][3]` 的形式表示 Data 段落，有效位为 0 的行留空，每个 Set 内的顺序无所谓

答案：

SetID	Tag	Data
0	0x06(000110)	`Array[1][15] ~ Array[2][1]`

SetID	Tag	Data
0	0x07(000111)	`Array[2][14] ~ Array[3][0]`
1	0x07(000111)	`Array[3][1] ~ Array[3][4]`
1	0x06(000110)	`Array[2][2] ~ Array[2][5]`

4.2

修改条件为

- 地址宽度为 10
- 数组的起始地址为 0b0010000000（即二进制表示）
- Cache 的容量为 16 Byte，Block size = 4 Byte，全相联
- 替换算法为 LRU

4.2.0

Tag 的位数是多少?

答案:

E = 4, S = 1, B = 4

t = m - s - b = 10 - 0 - 2 = 8

8 位

4.2.1

原始程序会发生几次缓存miss?

答案: 4

4.2.2

如果 LENGTH = 16，那么会发生几次缓存miss?

答案: 4

4.2.3

如果 LENGTH = 17，那么会发生几次缓存miss?

答案: 7

4.2.4

请画出在 LENGTH = 17 时，程序执行结束时的高速缓存状态，假设一开始全空。

> 可以用 `Array[0][0] ~ Array[0][3]` 的形式表示 Data 段落，有效位为 0 的行留空

SetID	Tag	Data
0	0x29(00101001)	`Array[2][2] ~ Array[2][5]`
0	0x2c(00101100)	`Array[2][14] ~ Array[3][0]`
0	0x2d(00101101)	`Array[3][1] ~ Array[3][4]`

SetID	Tag	Data
0	0x28(00101000)	`Array[1][15] ~ Array[2][1]`

解析：

有疑问的同学可以用 cachelab 的 csim 来验证自己的答案。

```
import subprocess

start = 0b0001000000
LENGTH = 8

with open("data.txt", "w") as f:
    for i in range(4):
        for j in range(4):
            addr = start + i * LENGTH + j
            f.write(f"S {hex(addr)},1 0\n")

output = subprocess.run(["./csim-ref",
                "-s", "2",
                "-E", "2",
                "-b", "2",
                "-t", "data.txt",
                "-v"
                ], capture_output=True)

print(output.stdout.decode())
```