Homework Week10

6.22

假设磁道密度为 t，即沿半径单位长度上的磁道数为 t.

盘面总半径为 r，圆洞半径为 xr，那么有磁道部分的半径为 $(1-x)r$，磁道数为 $(1-x)rt$.

圆洞周长为 $2\pi xr$，这是磁道上的位数.

总的位数是磁道数乘磁道上的位数，$2\pi xr(1-x)rt = 2\pi r^2t(1-x)x$.

它是关于 x 的二次函数，当 $x = \frac{1}{2}$ 时有最大值

6.23

访问时间 $=$ 寻道时间 $+$ 平均旋转时间 $+$ 传送时间

$$
\begin{aligned}
&= 4 + (\frac{1}{2} \times \frac{1}{RPM} + \frac{1}{800} \times \frac{1}{RPM}) \times 60 \times 1000 \text{ ms/min} \\
&= 4 + (\frac{1}{2} \times \frac{1}{15000} + \frac{1}{800} \times \frac{1}{15000}) \times 60 \times 1000 \text{ ms/min} = 4 + 2 + \frac{4}{800} \text{ ms/min} = 6.005 \text{ ms}
\end{aligned}
$$

6.24

$T_{avg\ seek} = 4 \text{ ms}, T_{avg\ rotation} = 1/2 \times 1/15000 \times 60 \times 1000 = 2\text{ms}$

1. 最好情况

所有逻辑块沿磁头读取方向顺序排列并且在同一个磁道上（磁道满情况除外），读完一个逻辑块后可以紧接着读下一个，

由于每扇区 512 B，总共 2 MB 是需要大约 4000 个扇区

平均每磁道 1000 扇区，故大约 4 磁道

总共 $2 \times 2 \times 4 + T_{avg\ seek} + T_{avg\ rotation} = 22$ ms

2. 随机情况

寻找每一个逻辑块的平均情况，是需要半个旋转周期以及平均寻道时间

半个旋转周期是 $T_{avg\ rotation}$

寻道时间 $T_{avg\ seek}$

总共要找 4000 个逻辑块

$4000 * (T_{avg\ seek} + T_{avg\ rotation}) = 24000 \text{ ms} = 24 \text{ s}$

6.25

$C = B \times E \times S,\ S = C/(B \times E)$

$b = \log_2(B)$

$s = \log_2(S)$

$t = m - b - s$

高速缓存	m	C	B	E	S	t	s	b
1.	32	1024	4	4	64	24	6	2
2.	32	1024	4	256	1	30	0	2
3.	32	1024	8	1	128	22	7	3
4.	32	1024	8	128	1	29	0	3
5.	32	1024	32	1	32	22	5	5
6.	32	1024	32	4	8	24	3	5

6.29

A.

bit 11 ... 4	bit 3...2	bit 1...0
CT	CI	CO

B.

`0x834 = 0b10000011 01 00`
`0x836 = 0b10000011 01 10`
`0xFFD = 0b11111111 11 01`

操作	地址	命中?	读出的值
读	0x834	不命中	未知
写	0x836	命中	未知
读	0xFFD	命中	0xC0

6.34

块大小为 `16` 字节，地址位为 `4` 位，`b == 4`
直接映射，每组 `1` 行
总共 `32` 字节，有两组，索引位为 `1` 位，`s == 1`
每个组恰好存数组中的一行

`src` 数组的地址从 `0` 到 `63`，其中
`src[0]` 和 `src[2]` 的地址的索引位为 `0`，它们使用缓存的组 `0`
`src[1]` 和 `src[3]` 的地址的索引位为 `1`，它们使用缓存的组 `1`
`dst` 数组的地址从 `64` 到 `127`，其中
`dst[0]` 和 `dst[2]` 的地址的索引位为 `0`，它们使用缓存的组 `0`
`dst[1]` 和 `dst[3]` 的地址的索引位为 `1`，它们使用缓存的组 `1`

故 `src[0]` `src[2]` `dst[0]` `dst[2]` 共享组 `0`
`src[1]` `src[3]` `dst[1]` `dst[3]` 共享组 `1`

`dst` 数组每次写都在不同行，其中对组 `0` 交替写 `dst[0]` 和第 `dst[2]` 行，对组 `1` 交替写第 `dst[1]` 行和第 `dst[3]` 行，每次写都必然不命中

对 `src[0][0]`，有冷不命中，其后每一行开始时，缓存中显然不会有这一行的数据，故每一行的第一次读都不命中

每次写 `dst` 时，会覆盖相应组，故写 `dst` 后，`src` 相应组会不命中，相反，另一组没有被覆盖，会命中

dst数组	列0	列1	列2	列3
行0	m	m	m	m
行1	m	m	m	m
行2	m	m	m	m
行3	m	m	m	m

src数组	列0	列1	列2	列3
行0	m	m	h	m
行1	m	h	m	h
行2	m	m	h	m
行3	m	h	m	h

6.36

A.
组数 `s == 32`，块偏移 `4` 位，组索引 `5` 位。缓存每一行可以存储数组的 `4` 个元素。
数组每一行 `128` 个 `int` 变量共 `512` 字节，所有的缓存恰好可以存储数组的一行
两行共 `1024` 字节，地址位 `10` 位，故而两行中的相同列放在缓存的同一组

函数中正是对两行中相同的每一列做读取，故每一次操作都会冲突不命中并覆盖对方，不命中率 `100%`

B.

组数为 `64`，组索引 `6` 位，全部缓存恰能完全存储数组，不命中只有每一组首次访问的冷不命中，不命中率 `25%`

C.

组数为 `s == 16`，组索引 `4` 位，地址中有两位进入标志位。
每组 `E == 2` 行，每隔 `64` 个元素放在同一组中，

`a[0][0...3] a[0][64...67] a[1][0...3] a[1][64...67]` 在同一组中，
`a[0][4...7] a[0][68...71] a[1][4...7] a[1][68...71]` 在另一组中，其它以此类推。

首先读 `a[0][0...3]` 和 `a[1][0...3]`，它们会分别发生一次不命中后放在这一组的不同行中，有三次命中
然后读 `a[0][4...7]` 和 `a[1][4...7]`，同上
直到 `a[0][60...63]` 和 `a[1][60...63]`

每一次读都有冷不命中，总的不命中率为 `25%`

在 `a[0][64...67]` 和 `a[1][64...67]` 时，它们会放第一次操作的相同组中，前者会发生一次不命中，丢弃当前最久未访问的缓存行并覆盖，再有三次命中。
对于后者，根据 LRU，前者覆盖的缓存行不是最久未访问的，所以会覆盖另一行，两者不会冲突。
故分别有一次不命中和三次命中，总的不命中率为 `25%`

对之后的操作同理。

总的不命中率为 `25%`

D.

不能，缓存中每个行都只能存放 `4` 个 `int` 变量，而对任何一行的初次访问至少发生一次不命中，不命中率不会低于 `25%`

E.

能，当块大小变大时，每个行能存储更多的变量，而不命中仍然只有首次发生，故保证不命中一次而命中更多次。不命中率下降。

6.41

结构体大小为 `4` 字节，缓存每行 `4` 字节，故缓存每行中是一个结构体对象，也就是二维数组中的一个元素。由于直接映射，所以每一组中恰好是一个元素。

循环中每次读完一个元素，对于 `buffer[i][j].r` 总是不命中，将这个元素覆盖缓存，之后对于 `buffer[i][j].g` `buffer[i][j].b` `buffer[i][j].a`，总是命中。故每个元素有一次不命中和三次命中，不命中率 `25%`。
每个元素都恰好占据缓存的一整行，所以每次写一个新的元素时都会发生一次不命中并覆盖对应缓存行，元素之间的关系对不命中率没有影响。

总的不命中率为 `25%`.