CSID #1	**W3h2b**
CSID #2	

Part 1: 1a

Place the letters CO, CI, or CT into each box indicating if the corresponding bit number is part of the cache block offset (CO), the cache set index (CI), or the cache tag (CT).

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CT	CT	CT	CT	CT	CT	CT	CT	CT	CI	CI	CI	CO	CO	CO

Part 1: 1b

For each row, fill in the cache set index, cache tag, and value with a hexadecimal number. In the Hit column, place an H if the address produces a hit and M if the address produces a miss.

Operation	Address	Cache Set Index	Tag	Hit	Value
Read	0x0117	2	004	M	unknown
Read	0x64C2	0	193	H	99
Read	0x7BFE	7	1EF	M	unknown
Read	0x0110	2	004	H	unknown

Part 3:

For each scenario, enter the miss rate as a percentage and then explain why the miss rate is what it is.

Scenario	Miss Rate	Explanation
A	25%	In sumA, function is reading data sequentially in row-major order in direct-mapped cache. And we will load 16 bytes in each cache block, which means cache will be loaded with 4 data from array in one read. Then, one miss will result next 3 hits. Also, we have 64 rows in each column. And 64 is multiple of 4, then each row has average miss rate of 25%. Then, this scenario will have 25% miss rate overall.
B	100%	In sumB, function is reading data in column-major order. Since we have 64 column each row, each row will have 64*4 = 256 bytes difference in address. When we have a[0][0] map to set 0,we access next data a[16][0] which has address 0x1000(or 16^3 bytes) higher than a[0][0]. Since we have 4bits for index and 7bits for index. a[16][0] will replace a[0][0], and when we access a[0][1] which is loaded with a[0][0], it will be a miss. Thus, it will never hit.
C	50%	In sumC, function iterates array in every 2 rows and 2 columns in column-major order. In this function, we will access 4 data in every iteration, which are a[i][j], a[i][j+1], a[i+1][j] and a[i+1][j+1]. When we have miss on a[i][j], we much have a hit on a[i][j+1]. Also, when we have miss on a[i+1][j], we mush have a hit on a[i+1][j+1]. Then, in one iteration we will have 50% miss rate. Overall, we will have 50% miss rate in this function.
D	25%	In sumB, function is reading data in column-major order with array of 68 rows and 68 columns. It is similar to scenario B, but it has 68*4=272 bytes difference in address to next row. Since 272 is not multiple of 16^3=4096, which means two different rows will not map to the same cache block based on index bits. Then, all rows will be map to distinct cache block. As a result, it will reading data in the way like scenario A, where we have 1 miss and result next 3 hit.

Part 4:

Enter your log entries in this table.

Log #	Average elapsed time	Time relative to baseline	Explanation
1	1986814.2	1	1976438.3
2	356628.3	0.180	Change column-major to row-major to improve spatial locality. Then, we will save loading time by decrease miss rate.
3	313543	0.158	Merge similar loop into one in order to save time from reread data and decrease compulsory miss, which decrease over all miss rate as well.
4	3044724	0.154	Merge similar write, where multiple write into same target, into one to save write time
5	294189.4	0.149	In the first loop, we save newlife into each cell instead of add into it. Then, we can remove initializing newlife which take waste time on writing to 0.
6	275312.8	0.140	Merge 3 loops into one and save read time and write time and merge all write into one to save write rime.
7	50663	0.025	Merge all loop into one and replace new world by old world and there is no write.
8			
9			
10			

11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			