CSE 5441
Harsh Gupta
(gupta.749@osu.edu)

Instructor: Dr. Jeffrey S. Jones
Grader: Karan Grover
Grader: Gregory Ochs

Question 1

a.

m	C	B	E	S	t	s	b
32	262144	256	4	256	16	8	8

b.

Size of each element: 8 bytes

Size of each cache line: 256 bytes

Number of elements in each cache line: 256/8 = 32 elements

Assuming cold miss

HIT RATIO for x and y

When we first try to access x[0], there will be a cache miss and then there will be 32 elements, starting from x[0] to x[31] which will be loaded in the cache which will go to set 0's line 0 (Here the line number is taken just for an instance)

Similarly when we try to access y[0], there will be a cache miss followed by 32 elements of y starting from y[0] through y[31] which will be loaded in the cache in set 0 and line 1 for instance. (Here there is no eviction for set 0 as we have more than one line per set which can accommodate other elements in different line of the same set)

The next 31 operation for x and y will lead to a cache hit.

When we will try to access the 32^{nd} element of x and y there will be a cache miss which will load the next 32 elements of x and y in two different lines of set 1. Again a cache miss lead to 31 cache hits.

This process will continue till all the elements of x and y are not loaded in the cache in different sets starting from set 0 through set 7.

In all these access, there is one cache miss followed by 31 cache hits.

Therefore,

hit ratio for x = 31/32 = 0.96875 = 96.875% (approximately 97%)

hit ratio for y = 31/32 = 0.96875 = 96.875% (approximately 97%)

HIT RATIO for a and b

When we first try to access a[0], there will be a cache miss and then there will be 32 elements, starting from a[0] to a[31] which will be loaded in the cache which will go to set 128's line 0 (Here the line number is taken just for an instance)

Similarly when we try to access b[0], there will be a cache miss followed by 32 elements of y starting from b[0] through b[31] which will be loaded in the cache in set 128 and line 1 for instance. (Here there is no eviction for set 128 as we have more than one line per set which can accommodate other elements in different line of the same set)

The next 31 operation for a and b will lead to a cache hit.

When we will try to access the 32nd element of a and b there will be a cache miss which will load the next 32 elements of a and b in two different lines of set 129. Again, a cache miss lead to 31 cache hits.

This process will continue till all the elements of a and b are not loaded in the cache in different sets starting from set 128 through set 143.

In all these access, there is one cache miss followed by 31 cache hits.

Therefore,

hit rate ratio for a = 31/32 **= 0.96875 = 96.875%** (approximately 97%)
hit rate ratio for b = 31/32 **= 0.96875 = 96.875%** (approximately 97%)
c. Hit rate ratios

x	96.875% (approximately 97%)
y	96.875% (approximately 97%)
a	96.875% (approximately 97%)
b	96.875% (approximately 97%)

The overall hit rate ration will also be 96.875% (approximately 97%) as all the arrays have the same hit rate ratio.

d. Assuming cold miss

Set	Lines	Value
0	0	x[0], x[1] … x[31]
	1	y[0], y[1] … y[31]
	2	Unknown value
	3	Unknown value

Set	Lines	Value
128	0	a[0], a[1] … a[31]
	1	b[0], b[1] … b[31]
	2	Unknown value
	3	Unknown value

All other sets between 0 to 128 will have unknown values.

e.

Set 0	Line 0	x[0], x[1], … x[31]
	Line 1	y[0], y[1], … y[31]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 1	Line 0	x[32], x[33], … x[63]
	Line 1	y[32], y[33], … y[63]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 2	Line 0	x[64], x[65], … x[95]
	Line 1	y[64], y[65], … y[95]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 3	Line 0	x[96], x[97], … x[127]
	Line 1	y[96], y[97], … y[127]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 4	Line 0	x[128], x[129], … x[159]
	Line 1	y[128], y[129], … y[159]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 5	Line 0	x[160], x[161], … x[191]
	Line 1	y[160], y[161], … y[191]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 6	Line 0	x[192], x[193], … x[223]
	Line 1	y[192], y[193], … y[223]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 7	Line 0	x[224], x[225], … x[255]
	Line 1	y[224], y[225], … y[255]
	Line 2	Unknown Value
	Line 3	Unknown Value

Set 128	Line 0	a[0], a[1], … a[31]
	Line 1	b[0], b[1], … b[31]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 129	Line 0	a[32], a[33], … a[63]
	Line 1	b[32], b[33], … b[63]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 130	Line 0	a[64], a[65], … a[95]
	Line 1	b[64], b[65], … b[95]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 131	Line 0	a[96], a[97], … a[127]
	Line 1	b[96], b[97], … b[127]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 132	Line 0	a[128], a[129], … a[159]
	Line 1	b[128], b[129], … b[159]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 133	Line 0	a[160], a[161], … a[191]
	Line 1	b[160], b[161], … b[191]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 134	Line 0	a[192], a[193], … a[223]
	Line 1	b[192], b[193], … b[223]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 135	Line 0	a[224], a[225], … a[255]
	Line 1	b[224], b[225], … b[255]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 136	Line 0	a[256], a[257], … a[287]
	Line 1	b[256], b[257], … b[287]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 137	Line 0	a[288], a[289], … a[319]
	Line 1	b[288], b[289], … b[319]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 138	Line 0	a[320], a[321], … a[351]
	Line 1	b[320], b[321], … b[351]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 139	Line 0	a[352], a[353], … a[383]
	Line 1	b[352], b[353], … b[383]
	Line 2	Unknown Value
	Line 3	Unknown Value

Set 140	Line 0	a[384], a[385], … a[415]
	Line 1	b[384], b[385], … b[415]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 141	Line 0	a[416], a[417], … a[447]
	Line 1	b[416], b[417], … b[447]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 142	Line 0	a[448], a[449], … a[479]
	Line 1	b[448], b[449], … b[479]
	Line 2	Unknown Value
	Line 3	Unknown Value
Set 143	Line 0	a[480], a[481], … a[511]
	Line 1	b[480], b[481], … b[511]
	Line 2	Unknown Value
	Line 3	Unknown Value

Question 2

Given
E = 8
Eviction bits = 3

Assume
s = 0
Each line holds just 1 element of size 1

Let us use this eviction bits to implement a FIFO and use the three eviction bits to determine all the eight different cache lines. The way we do it is to have the base value of decimal 0 i.e binary 000 against each cache line and then increment the value of each cache line by decimal 1 plus the value of the cache line above it till we reach decimal 7 i.e binary 111. Once we reach the maximum value of decimal 7 we set the value of eviction bits in each cache line to decimal 0. In this way we follow a linear sequence of eviction and each line gets evicted in a first in first out fashion, but each cache line gets the privilege of staying with the same value for 7 strides.

Let us try to understand this with a scenario. Suppose currently all the 8 cache lines has some value and there is a cache miss. What we do is reset eviction bits for all the cache lines and we move to line 0 i.e binary 000 to evict and replace it with the current value. Now we will increment the eviction bit for line 0 to 001, so the next time if there is a cache miss it will see eviction bit of line 0 which has the value 001 in it and hence we will evict values of line 1. Once we evict line 1, we will set the eviction bit of line 1 to 010 so that it points to line 2. In this way, we will keep moving forward.

Address access patterns

Policy would perform well		Policy would perform poorly	
Addresses which are accessed lately are also accessed frequently in the future		Addresses which are accessed lately are not accessed frequently in the future	
AA00	Miss	AA00	Miss
AA01	Miss	AA01	Miss
AA02	Miss	AA02	Miss
AA03	Miss	AA03	Miss
AA04	Miss	AA04	Miss
AA05	Miss	AA05	Miss
AA06	Miss	AA06	Miss
AA07	Miss	AA07	Miss
AA06	Hit	AA08	Miss, replace AA00
AA06	Hit	AA00	Miss, replace AA01
AA08	Miss, replace AA00	AA01	Miss, replace AA02
AA07	Hit	AA02	Miss, replace AA03
AA07	Hit	AA03	Miss, replace AA04
AA0A	Miss replace AA01	AA04	Miss, replace AA05

Question 3

Including the initial scan to determine that a cache miss has occurred, we need to scan 3 times the cache lines in the affected set to successfully replace a cache line requiring eviction.

Information determined by each scan and what is considered when each of the cache line is scanned.

Scan 1	To find out that a cache miss has happened. **Inspect the valid bit of the cache line. If the valid bit is correct then inspect the tag bits.**
Scan 2	Scan all the lines to check if there is an empty line. **Inspect the valid bit to see if the cache line is empty or not.**
Scan 3	If there is no empty line found in the scan above, then determine a cache line which has to be evicted. **Inspect the valid bit and eviction bit, to determine which cache line needs to be evicted.**

Once we have the eviction bits with us, we must scan again to go to the line with the known eviction bits and then replace the element with the memory block we want to have in cache. Sometimes this scan is also considered a proper scan, in which case we can say that there are four scans in total.

Question 4

Assumption: The question didn't ask for an example of pattern. You were supposed to give th

E = 4
s = 0
b = 1
(sizeOf) element = 2 i.e each line can hold only 1 element

<table>
<tr><td>LRU cache</td><td>LFU cache</td></tr>
<tr><td>Eviction Policy: Evict cache line whose last access time is the earliest among all</td><td>Eviction Policy: Evict cache line whose access frequency is the least</td></tr>
<tr><td>Address access pattern
<table><tr><td>AA05</td><td>Miss (Write to Line 0)</td></tr><tr><td>AA06</td><td>Miss (Write to Line 1)</td></tr><tr><td>AA07</td><td>Miss (Write to Line 2)</td></tr><tr><td>AA08</td><td>Miss (Write to Line 3)</td></tr><tr><td>AA06</td><td>Hit</td></tr><tr><td>AB07</td><td>Miss (Write to Line 0 replacing AA05)</td></tr><tr><td>AB07</td><td>Hit</td></tr><tr><td>AB08</td><td>Miss (Write to Line 2 replacing AA07)</td></tr><tr><td>AA06</td><td>Hit</td></tr><tr><td>AB07</td><td>Hit</td></tr></table></td><td>Address access pattern
<table><tr><td>AA05</td><td>Miss (Write to Line 0)</td></tr><tr><td>AA06</td><td>Miss (Write to Line 1)</td></tr><tr><td>AA07</td><td>Miss (Write to Line 2)</td></tr><tr><td>AA08</td><td>Miss (Write to Line 3)</td></tr><tr><td>AA06</td><td>Hit</td></tr><tr><td>AA08</td><td>Hit</td></tr><tr><td>AA06</td><td>Hit</td></tr><tr><td>AB05</td><td>Miss (Write to Line 0 replacing AA05)</td></tr><tr><td>AB06</td><td>Miss (Write to Line 2 replacing AA07)</td></tr><tr><td>AB05</td><td>Hit</td></tr><tr><td>AA06</td><td>Hit</td></tr><tr><td>AA08</td><td>Hit</td></tr></table></td></tr>
<tr><td>LRU performs better if the elements which were accessed in the past are also accessed currently.</td><td>LFU performs well when addresses which were accessed in the past, are getting accessed frequently currently.</td></tr>
</table>

Question 5

a.

m	C	B	E	S	t	s	b
32	65536	256	256	1	24	0	8

b.

Element size = 8 bytes
Block size = 256 bytes
Number of elements = 256/8 = 32

To access one element from x or y there is one miss which will load 32 elements in the cache line. Access of next 31 elements will result in a cache hit.

Therefore, hit rate ratio of x is 31/32 = 96.875 % = **97 % (approx.)**

Therefore, hit rate ratio of y is 31/32 = 96.875 % = **97 % (approx.)**

c. As individual hit rate of x and y are 96.875% each, hence the overall hit ratio is also 96.875% or approx. 97%.

d. Cache contents after execution of I loop is assuming values go linearly in the cache lines

Line 0	x[4096],x[4097],……..x[4127] y[4096],y[4097],……..y[4127]
Line 1	x[4128],x[4129],……..x[4159] y[4128],y[4129],……..y[4159]
..	
..	
..	
Line 254	x[8128],x[8129],……..x[8159] y[8128],y[8129],……..y[8159]
Line 255	x[8160],x[8161],……..x[8191] y[8160],y[8161],……..y[8191]

Question 6

$AMAT_A = 2 + (1-0.25) * AMAT_2$

$= 2 + (1-0.25) * (10 + (1-0.8)*AMAT_3)$

$= 2 + (1-0.25) * (10 + (1-0.8)*(80 + (1-0.95)*500))$

$= 25.25$

$AMAT_B = 2 + (1-0.5) * AMAT_2$

$= 2 + (1-0.5) * (5 + (1-0.65)*AMAT_3)$

$= 2 + (1-0.5) * (5 + (1-0.65)*(20 + (1-0.8)*500))$

$= 25.5$

As average memory access time of A is less than that of B, hence A will provide better AMAT