Homework 2 – cache part 2
Zewen Hua

1.

a.

m	C	B	E	S	t	s	b
32	2^{18}	256	4	256	16	8	8

b.

B/|element| = 32

ASK	GET	SET	MISS/HIT
x[0]	x[0-31]	Set0	M
y[0]	y[0-31]	Set0	M
a[0]	a[0-31]	Set128	M
b[0]	b[0-31]	Set128	M
a[1]			H
b[1]			H
x[1]			H
y[1]			H
a[2]			H
b[2]			H
a[3]			H
b[3]			H

hit_rate_x = 31/32

hit_rate_y = 31/32

hit_rate_a = 31/32

hit_rate_b = 31/32

c.

hit_rate_overall = (31+31+31*2*2)/(32*2+32*2*2) = 31/32

d.

As the table shows in question b:

Set0	x[0-31]
	y[0-31]
Set128	a[0-31]
	b[0-31]
Set129	a[32-63]
	b[32-63]
Other sets are empty!	

e.
b*S = 256 * 8 = 2048 > (255*2 + 1 + 1) > 256. So, there won't be any eviction.

Set0	x[0-31]
	y[0-31]
Set1	x[32-63]
	y[32-63]
…	…
Set7	x[224-255]
	y[224-255]
Set8 – Set127	EMPTY
Set128	a[0-31]
	b[0-31]
Set129	a[32-63]
	b[32-63]
…	…
Set143	a[480-511]
	b[480-511]
Set144-255	EMPTY

2.
It is an 8-way set-associative cache and the eviction bits are associated with each set instead of cache lines. I will use the eviction bits to record which line is going to be evicted next. And the eviction policy is "First in first out." It works like a queue: if the set is full, the first line will be evicted by the ninth line and the second line will be evicted by the tenth line …

This policy is very simple. So, if the first several data will be used few times or even once, this policy will work well than other complex ones. But if the data will be used many times, especially the first several pieces which will be evicted very early, the policy will perform very poorly. And the examples are shown below:

m	**C**	**B**	**E**	**S**	**t**	**s**	**b**
16	16	4	4	1	14	0	2

sizeof(int) = 1;
@a[20] = AA00;
"sum" in the register

Address pattern 1:
code: sum = a[0] + a[4] + a[8] + a[12] + a[16] + a[9] + a[8] + a[13] + a[9] + a[8] + a[13]

ASK	GET	MISS/HIT
a[0]	a[0-3]	M
a[4]	a[4-7]	M
a[8]	a[8-11]	M
a[12]	a[12-15]	M

a[16]	a[16-19]	M
a[9]		H
a[8]		H
a[13]		H
a[9]		H
a[8]		H
a[13]		H

As shown, when we load a[16-19], a[0-3] will be evicted;
So, hit_rate = 6/11 = 0.5
Then, it performs well in this pattern.

Address pattern 2:
code: sum = a[0] + a[4] + a[8] + a[12] + a[16] + a[0] + a[4] + a[8] + a[12] + a[16]

ASK	GET	MISS/HIT
a[0]	a[0-3]	M
a[4]	a[4-7]	M
a[8]	a[8-11]	M
a[12]	a[12-15]	M
a[16]	a[16-19]	M
a[0]	a[0-3]	M
a[4]	a[4-7]	M
a[8]	a[8-11]	M
a[12]	a[12-15]	M
a[16]	a[16-19]	M

As shown, hit_rate = 0
Then, it performs very poorly in this pattern.

3.
Including all kinds of scans, there will be 3 times for each cache line.

1) The address gets a “miss” or “hit”
2) If “miss”, check that it still has empty lines or not
3) If full, check which line to evict

1) valid bit, tag bits, cache block
2) valid bit
3) eviction bit

4.

m	C	B	E	S	t	s	b
16	16	4	4	1	14	0	2

sizeof(int) = 1;
@a[20] = AA00;
“sum” in the register

Address pattern 1:
code: sum = a[0] + a[1] + a[2] + a[4] + a[5] + a[8] + a[12] + a[13] + a[16] + a[13]

ASK	GET	MISS/HIT
a[0]	a[0-3]	M
a[1]		H
a[2]		H
a[4]	a[4-7]	M
a[5]		H
a[8]	a[8-11]	M
a[12]	a[12-15]	M
a[13]		H
a[16]	a[16-19]	M
a[14](LRU)		H
a[14](LFU)	a[12-15]	M

As shown, if LRU: when we load a[16-19], a[0-3] will be evicted;
if LFU: when we load a[16-19], a[12-15] will be evicted.
So, hit_rate_LRU = 5/10 = 0.5; hit_rate_LFU = 4/10 = 0.4
Then, LRU works better than LFU in this pattern.

Address pattern 2:
code: sum = a[0] + a[1] + a[2] + a[4] + a[5] + a[8] + a[9] + a[12] + a[16] + a[3]

ASK	GET	MISS/HIT
a[0]	a[0-3]	M
a[1]		H
a[2]		H
a[4]	a[4-7]	M
a[5]		H
a[8]	a[8-11]	M
a[9]		H
a[12]	a[12-15]	M
a[16]	a[16-19]	M
a[3](LRU)	a[0-3]	M
a[3](LFU)		H

As shown, if LRU: when we load a[16-19], a[0-3] will be evicted;
if LFU: when we load a[16-19], a[12-15] will be evicted.

So, hit_rate_LRU: 4/10 = 0.4; hit_rate_LFU: 5/10 = 0.5
Then, LFU works better than LRU in this pattern.

5.
a.

m	C	B	E	S	t	s	b
32	2^{16}	256	256	1	24	0	8

b.
B/|element| = 32
hit_rate_x = hit_rate_y = 31/32
c.
hit_overall = hit_rate_x = hit_rate_y = 31/32
d.
256*256/8/2 = 4096; 8192/4096 = 2; So,

Set0	x[4096-4127]
	y[4096-4127]
	x[4128-4159]
	y[4128-4159]
	…
	x[8160-8191]
	y[8160-8191]

6.
For A:
$AMAT_1 = 2 + (1 - 0.25)(AMAT_2)$
$= 2 + (1 - 0.25)(10 + (1 – 80\%)(AMAT_3))$
$= 2 + (1 - 0.25)(10 + (1 – 80\%)(80 + (1 – 0.95)(AMAT_m)))$
$= 2 + (1 - 0.25)(10 + (1 – 80\%)(80 + (1 – 0.95)(500)))$
$= 25.25$
For B:
$AMAT_1 = 2 + (1 - 0.5)(AMAT_2)$
$= 2 + (1 - 0.5)(5 + (1 – 65\%)(AMAT_3))$
$= 2 + (1 - 0.5)(5 + (1 – 65\%)(20 + (1 – 0.8)(AMAT_m)))$
$= 2 + (1 - 0.5)(5 + (1 – 65\%)(20 + (1 – 0.8)(500)))$
$= 25.5$

$AMAT_1\_A < AMAT_1\_B$. So, A is better.