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

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

Question 1

a.

Cache	m	C	B	E	S	t	s	b
C1	16	64	4	1	16	10	4	2
C2	16	64	16	1	4	10	2	4

b.

Address	C_1	C_2
BA00	Miss	Miss
BA04	Miss	Hit
AA08	Miss	Miss
BA05	Hit	Miss
AA14	Miss	Miss
AA11	Miss	Hit
AA13	Hit	Hit
AA38	Miss	Miss
AA09	Hit	Miss
AA0B	Hit	Hit
BA04	Hit	Miss
AA2B	Miss	Miss
BA05	Hit	Hit
BA06	Hit	Hit
AA09	Hit	Miss
AA11	Hit	Hit

For Cache 1 the final data content is

Set/Line	Address loaded
0	BA00-BA03
1	BA04-BA07
2	AA08-AA0B
3	Unknown Value
4	AA10-AA13
5	AA14-AA17
6	Unknown Value
7	Unknown Value
8	Unknown Value
9	Unknown Value
10	AA28-AA2B
11	Unknown Value
12	Unknown Value

13	Unknown Value
14	AA38-AA3B
15	Unknown Value

For cache 2 the final data content is

Set/Line	Address loaded
0	AA00-AA0F
1	AA10-AA1F
2	AA20 – AA2F
3	AA30-AA3F

c. For the given sequence of read addresses below, we will have more misses for cache 2 than cache 1. We see that the hit ration for cache 1 is 50% but for cache 2 is 0%

Address	C_1	C_2
BA0D	Miss	Miss
AA02	Miss	Miss
BA0E	Hit	Miss
AA00	Hit	Miss

Question 2

a. Address bits of the processor = t + s + b bits = 22 + 8 + 4 = 34 bits

Therefore, address space = 2^{34}

b. This is a direct mapped cache, hence E = 1

usable cache size C = S * B * E bytes = $2^8 * 2^4 * 1$ bytes = 2^{12} bytes = 4096 bytes = 32768 bits

c. From above we can see usable cache size is 12 bits, but from part a we see that there are 22 tag bits, 8 set bits, and we have 4 bits block offset as well. So making 1 valid bit into consideration the total number of bits will be the sum of tag bits, offset bits and valid bit. Therefore, the number of bits to implement cache is 22+8+4+1 = **35.**

d.

Tag	Set	Offset
22	8	4

Each cache has 22 tag bits associated with it, hence if any one of these 22 bits is different, it will fall in the same set. Therefore, there are **2^{22} distinct memory blocks** which will use the same cache line.

Question 3

The pseudo code to find the minimum valued integer

Let there be an array A of size 1,000,000 integers of size 4 bytes each

```
If A.length == 0 then
        <Print error message for invalid input>
End if
minValue : A[0]
for all i from 1 to A.length do
        if A[i] < minValue then
                minValue = A[i]
        end if
end for
```

Given Size of int = 4 bytes

Assume: Cache line size = N*sizeOf(int)

Array is stored in contiguous memory location in row major format

As we are accessing the array elements contiguously, hence we will have a cache miss to access the first element in the cache line, but the next N-1 elements will be already there resulting in a cache hit. Therefore, we have a cache hit rate of (N-1)/N. According to the given problem we should have a hit rate of at least 87.5%.

{(N-1)/N}*100 >= 87.5

Or, N >= 8

As sizeOf(int) = 4, we should have a cache line of at least 8*4 bytes i.e 32 bytes. The number of cache lines per set and number of sets does not have any effect on the cache hit ration, hence it can be anything starting from its minimum value.

m	**C**	**B**	**E**	**S**	**t**	**s**	**b**
64	**>= 32**	**>= 32**	**>= 1**	**>= 1**	**<= 59**	**>= 0**	**>= 5**

m	**C**	**B**	**E**	**S**	**t**	**s**	**b**
64	**32**	**32**	**1**	**1**	**59**	**0**	**5**

Question 4

Case 1	Case 2

Let there be a 2D array A of size M*N
If A.length == 0 or A[0].length == 0
<Print error message for invalid input>
End if
Sum : 0
for all i from 1 to A.length do
for all j from 1 to A[0].length do
sum = sum + A[i][j]
end for
Avg = sum/(A.length + A[0].length)
	Let there be a 2D array A of size M*N
	If A.length == 0 or A[0].length == 0
	<Print error message for invalid input>
	End if
	Sum : 0
	for all j from 1 to A[0].length do
	for all i from 1 to A.length do
	sum = sum + A[i][j]
	end for
	Avg = sum/(A.length + A[0].length)

In case 1 we are accessing the elements row wise, where we are accessing each element of the row with an access stride of 1 starting from the first element of the row, and then moving on to the next row. In case 2 we are accessing the elements column wise, where we are accessing each element of the column with an access stride of 1 starting from the first element of the first column, and then moving on to the next column when all elements in the respective column is already accessed.

C language uses row major to store elements of the array

In case 1, we will have a good locality of reference contrasting to **case 2** which has bad locality of reference. The difference between the two cases is the access stride they are taking. While in the former case the access stride is of 1 element, where as in the later case, it is of N elements. In case 2 for each access there will be a cache miss but in case 1, there will be a cache miss only for the first element to be access in the cache line, for all the other elements present in the cache we will have cache hit. If the cache line can hold a total of X elements, then hit ration for case 1 will be (X-1)/X where as the hit ratio for case 2 will be 0.

If we write the same code in **Fortran which uses column major**, the code in case 2 will perform better than case 1. Case 2 here will have a hit ratio of (X-1)/X, where as case 1 will have a hit ratio of 0.

Question 5

What remains in the cache after completion of the extended for loop is

Set	Elements	
0	A[2046-2047]	Other unknown Value
1	A[1026-1029]	
2	A[1030-1033]	
3	A[1034-1037]	
⋮	⋮	
253	A[2034-2037]	
254	A[2038-2041]	
255	A[2042-2045]	

- For accessing the first two elements there will be one ram access and one cache miss and one cache hit.
- For the next 2044 elements, there will be one ram access for one cache miss and three cache hits.
- For the last two elements, there will be one ram access and one cache miss and one cache hit.

For a large part of memory access there is one miss for the next three hits, hence the approximate hit rate ratio percentage is 75%. Total memory access is 1+(2044/4)+1 = 1+511+1 = **513**