Homework 1

Tarun Ronur Sasikumar

1.

Given,
Size of Address: 32 bits
Cache Capacity **(C)**: 256KBytes
Block Size**(B):** 128 Bytes

Let:
No. of tag bits = **t**
No. of set-index bits = **s**
No. of block offset bits = **b**

I) Direct Mapped Cache:

- Total number of lines in the cache:
  Total Capacity: 256 Kbytes
  Block Size: 128 Bytes
  => Total number of lines = Total Capacity / Block Size
  => (256 x 1024) / 128 = 2048
  => **2048**

- Number of cache sets:
  Associativity: E = 1 (Since it is a direct mapped cache)
  C = E x S x B
  S = C / (E x B)
  S = (256 Kbytes)/(1 x 128 Bytes)
  S = (256 x 1024)/128
  S = 2048

- Value of b:
  Block Size = 128
  => b = $\text{Log}_2(128) = 7$
  => **7**

- Value of s:
  Number of Sets = 2048
  => s = Log 2 (2048) = 11
  => **11**

- Value of t:
  Size of address = 32bits
  s = 11 bits
  b = 7 bits
  => t = 32 – (11 + 7)
  => **14**

II) 4-way Set associative:

- Total number of lines in the cache:
  Total Capacity: 256 Kbytes
  Block Size: 128 Bytes
  => Total number of lines = Total Capacity / Block Size
  => (256 x 1024) / 128 = 2048
  => **2048**

- Number of cache sets:
  Associativity: E = 4
  C = E x S x B
  S = C / (E x B)
  S = (256 Kbytes)/(4 x 128 Bytes)
  S = (256 x 1024)/(4 x 128)
  S = 512

- Value of b:
  Block Size = 128
  => b = $\text{Log}_2(128) = 7$
  => **7**

- Value of s:
  Number of Sets = 512
  => s = $\text{Log}_2(512) = 9$
  => **9**

- Value of t:
  Size of address = 32bits
  s = 9 bits
  b = 7 bits
  => t = 32 – (9 + 7)
  => **16**

III) Fully associative:

- Total number of lines in the cache:
  Total Capacity: 256 Kbytes
  Block Size: 128 Bytes
  => Total number of lines = Total Capacity / Block Size
  => (256 x 1024) / 128 = 2048
  => **2048**

- Number of cache sets:
  Number of Sets: S = 1 (Since it is fully associative)

- Value of b:
  Block Size = 128
  => b = $\text{Log}_2(128) = 7$
  => **7**

- Value of s:

Number of Sets = 1
=> s = $\log_2(1) = 0$
=> **0**

- Value of t:
  Size of address = 32bits
  s = 0 bits
  b = 7 bits
  => t = 32 – (0 + 7)
  => **25**

2)

```
#define size 4096
int A[size];
s = 0;
for (it=0; it < stride; it++)
        for (i=0; i<size; i += stride)
                s +=A[i];
```

Given:
Associativity (**E**) : 1 (Since it is a direct mapped cache)
Capacity (**C**) : 16 Kbytes
LineSize (**B**) : 64 Bytes
WordSize for "*int*" (**w**) : 4 Bytes

Number of Sets => Number of Lines = Capacity / LineSize
= 16 Kbytes / 64 Bytes
= (16 x 1024) / 64
= 256

(a) Size = 4096. Stride = 1,4,16,2048,8192
Number of Bytes occupied by A => 4096 * 4 => 16384 Bytes => 16 Kbytes.
Since the Size of the Array is equal to the total cache capacity, and the associativity is 1, the entire Array can be stored into the cache without evicting a previously stored Block.

- Stride = 1
  The loop reduces to just the inner loop, with a stride of 1.
  => Number of misses will be equal to the Total Number of lines, since there will be one **Miss** per line(*cold miss*) and the rest of the Words in the Line/Block will be a **Hit**
  => No. of Misses = 256

- Stride = 4
  Since the inner loop is accessing all the elements in strides of 4, the case reduces to the same case as above. Although the loop might be accessing only 1/4th the number of elements, it is still going to load the same number of Lines into the cache(as it is still accessing the first element of each line). Hence the number of misses will be the same the previous case.
  => No. of Misses = 256

- Stride = 16
  Since the inner loop is accessing all the elements in strides of 16, the case reduces to the same case as above. Although the loop might be accessing only 1/16th the number of elements, it is still going to load the same number of Lines into the cache . Hence the number of misses will

be the same the previous case.
=> No. of Misses = 256

- Stride = 2048
  The inner loop in this case only accesses the elements A[0] and A[2048], which are both a cache-miss the first time as they are in different blocks. Every iteration of the outer loop still only accesses the same 2 elements.
  => No. of Misses = 2

- Stride = 8192
  In this case the inner loop only accesses one element A[0]. For every iteration of the outer loop, the only element being accessed is A[0].
  => No. of misses = 1

b) Size = 8192. Stride = 1,4,16,2048,8192
Number of Bytes occupied by A => 8192 * 4 => 32768 Bytes => 32 Kbytes.
Number of Lines in the Cache => 256 (From the previous result)
Number of lines required to store the entire Array = 512

- Stride = 1
  The loop reduces to just the inner loop, with a stride of 1.
  => Number of misses will be equal to the Total Number of lines required to store the entire array. There will be one **Miss** per line(*cold miss*) and the rest of the Words in the Line/Block will be a **Hit.** (Note: Once the cache is full (256 Lines of A), subsequent accesses will cause for eviction of a previously present line**)**
  => No. of Misses = 512

- Stride = 4
  Since the inner loop is accessing all the elements in strides of 4, the case reduces to the same case as above. Although the loop might be accessing only 1/4th the number of elements, it is still going to load the same number of Lines into the cache(as it is still accessing the first element of each line). Hence the number of misses will be the same as the previous case * number of iterations of outer loop.
  => No. of Misses = 512 * 4 = 2048

- Stride = 16
  Since the inner loop is accessing all the elements in strides of 16, the case reduces to the same case as above. Although the loop might be accessing only 1/16th the number of elements, it is still going to load the same number of Lines into the cache . Hence the number of misses will be the same as the previous case * number of iterations of outer loop.
  => No. of Misses = 512 * 16 = 8192

- Stride = 2048
  The inner loop in this case only accesses the elements A[0], A[2048], A[4096], A[6144] which are all a cache-miss the first time as they are in different blocks. A[4096] will evict A[0] as they map to the same block. Similarly to A[6144] evicts A[2048]
  Every iteration of the outer loop will cause 4 misses each time.
  => No. of Misses = 4 * 2048 = 8192

- Stride = 8192
  In this case the inner loop only accesses one element A[0]. For every iteration of the outer loop,

the only element being accessed is A[0].
=> No. of misses = 1

3) Given:
Associativity **(E) :** 4
Block Size **(B)** : B Bytes (32 <= B <= 1024 or $2^5 <= B <= 2^{10}$)
Cache Size (**C**) : 128 KBytes = 128 * 1024 Bytes
Word size for *int* **(w) :** 4
Access time (per **w):** t_l
Miss Penalty (per line): t_m

No. of lines in cache: Cache size/Block size => (128 * 1024) / B
No. of sets in cache: Cache size / (Block size * associativity) => C / (B *E) => 128 KB/ (4 * B)
No. of words per line: Block size / Word Size => B/w = B/4

a) size = 4 * 1024, stride = 1

size = 4096 words = 16 Kbytes
Line size = B bytes
No. of words per Line = B/4
No. of lines required to store **A =** Size of A / Size of a Block => 16 Kbytes / B

Since the elements in the array are accessed one at a time and every element is accessed once, no. of misses will be equal to the number of lines required to store the array.

No. of misses = No. of lines required **=> 16 Kbytes / B**

Once a line is loaded into the cache (after a cache-miss), the rest of the words in the Line are a cache hit

No. of hits = Total size of the Array (in words) - Total number of misses.
= **4096 – (16 Kbytes / B)**

Therefore,
Total time required for execution = (Total no. of hits * access time) + (Total no. of misses * (miss penalty + access time))

=> [t_l * (4096 - (16 Kbytes / B))] + [($t_l + t_m$) * (16Kbytes / B)]
=> $t_l(2^{12} – 2^{14}/B) + (t_l+t_m)(2^{14}/B)$

=> $(2^{14}/B)*t_l*(B – 4) + (2^{14}/B)*(t_l + t_m)$

b) size = 4 * 1024, stride = 16
No. of lines required to store complete the entire Array => Array size in bytes / Block Size
=> (4 * 4096) / B
where 32 <= B <= 1024

Since stride is 16, elements being accessed are of the order A[0], A[16], A[32] …
This can also be written as A[0], A[0] + 64 (Address in byte address)...
Hence consecutive accesses are 64 Bytes apart

We know that the block size B is between 32 and 1024 (inclusive)

This problem can therefore be split into 2 general solutions. When B <=64 and when B > 64

- When B = 32 or 64
  In this special case, the Block size is less than that required to store any number of consecutive accesses.
  Hence in this case every access will come from a separate line.

  This implies that the:
  No. of misses = No. of lines required = No. of accesses in the inner loop.

  No. of misses => 4096/16 = **256**

  No. of Hits => 15 * (No. of accesses in the inner loop) => **15 * 256**
  [Since every access after the first iteration of the outer loop is a cache-hit]

  Therefore,
  Total time required for execution = (Total no. of hits * access time) + (Total no. of misses * (miss penalty + access time))

  => $(15 * 256)\, t_l + 256(t_l + t_m)$
  => $\mathbf{256(16t_l + t_m)}$

- When B > 64
  No. of lines required = 4096/ (B / 4) => $2^{14}/B$

  No. of Misses = No. of line accesses => $\mathbf{2^{14}/B}$

  No. of Hits in the inner Loop = No. of lines * (No. of elements accessed per line - 1)
  => $\mathbf{((2^{14}/B) * ((B/4) * (1/16) - 1)}$
  => $(2^{14}/B * (B/2^6 - 1))$

  Total No. of hits = (15 * No. of Misses in the first iteration of the inner loop.)
  + (16 * No. of Hits in the first iteration of the inner loop.)
  => $(15 * 2^{14}/B) + (16 * (2^{14}/B * (B/2^6 - 1)))$
  => $(15 * 2^{14}/B) + (16 *(2^8 - 2^{14}/B))$
  => $(15 * 2^{14}/B) + 2^{12} - 2^{18}/B$

  Total time required for execution = (Total no. of hits * access time) + (Total no. of misses * (miss penalty + access time))

  => $(t_l * (15 * 2^{14}/B) + 2^{12} - 2^{18}/B) + (2^{14}/B *(t_l + t_m))$

  Simplifying the above we get
  => $\mathbf{2^{12} (t_l + 4t_m/B)}$

Before evaluating the next few cases let us derive a general formula for total time for execution.

Let:
N_m = Total no. of Misses
N = Total no. of accesses
t_l = access time
t_m = miss penalty

Total time = $(t_m + t_l)$ * (Total No. of Misses) + t_l (Total accesses – Total No. of misses)
=> $(t_m + t_l)(N_m) + t_l(N - N_m)$

Total time for execution => $N_m t_m + N t_l$

c) size = 64 * 1024, stride = 1

No. of line accesses => (64* 1024)/B
=> $2^{6+10}/B$
=> $2^{16}/B$

No. of misses in inner loop = No. of line accesses = **$2^{16}/B$**

Therefore total number of misses =>**16 * $2^{16}/B = 2^{20}/B$**

Total accesses => Total no. of misses => **$2^{20}/B$** (Every element being accessed cause a cache-miss)

Total time of execution => $N_m t_m + N t_l$
=> ($2^{20}/B$ * tm)+ ($2^{20}/B$ * t1)
=> ($2^{20}/B$) * $(t_l + t_m)$

d) size = 64 * 1024, stride = 16

No. of accesses for first iteration => Total size / stride => (64 * 1024) / 16 => 4096 => 2^{12}

Now, just as in case b) above, we need to handle two cases:

- When B = 32 or 64
  In this case, the Block size is less than that required to store any number of consecutive accesses.
  Hence in this case every access will come from a separate line.
  => every reference will be a miss
  No. of misses => 2^{12}
  In the outer loop, every iteration will replace existing elements in the cache. Hence, all *cache-misses*

  Therefore, total no. of misses =>
  16 * $2^{12} = 2^{16}$

  => **Total time required for execution = 2^{16} ($t_l + t_m$)**

- When B > 64
  No. of lines required => (64 * 1024) / (B * 4) => 2^{18}/B
  Therefore, No. of misses in one iteration => **2^{18}/B**

  Now, the outer loop will replace existing elements. Hence all cache-misses

  Therefore, Total no. of misses of all iterations = 16 * 2^{18}/B = **2^{22}/B**

  Total no. of accesses => No. of iterations * No. of lines required * No. of elements accessed per line.

  => 16 * (2^{18} / B) * (B / (4 * 16))
  => **2^{16}**

  Therefore using the formula derived above, we get:
  Total time required for execution = **$N_m t_m + N t_l$**
  => (2^{22}/B * t_m) + (2^{16} * t_l)

e) size = 64 * 1024, stride = 4 * 1024

No of references = size / stride => (64 * 1024) / (4 * 1024)
The elements being accessed will be:

A[0], A[4 * 1024], A[8 * 1024], A[12 * 1024] ….A[28 * 1024], A[32 * 1024]... A[60 * 1024]

All the elements of these all map to only 2 of the sets.

Considering the first 8 elements, as they all map to 2 sets and with an associativity of 4, the cache will have to evict these elements to accommodate the last 8 elements.

=> No. of misses per iteration = No. of references = 16

Total No. of misses from 4 * 1024 iterations => 16 * 4 * 1024

Total time required for execution => (2^2 * 2^4 * 2^{10}) (t_l + t_m)
=> 2^{16} (t_l + t_m)

f) size = 64 * 1024, stride = 16 * 1024

In this case, only 4 elements of the array are accessed:
A [0] , A[16 * 1024], A[32 * 1024], A[48 * 1024]

In the inner loop, all these for elements cause a cold miss.
Therefore, misses in the inner loop = 4
The outer loop, will not cause any more misses since the same elements are being accessed and they will be present in the cache.

No. of Hits = (stride – 1) * no. of misses in the first iteration.
= ((64 * 1024) -1) * 4
= (2^{16}– 1)* 4
Total time required for execution = (Total no. of hits * access time) + (Total no. of misses * (miss penalty + access time))

=> $((2^{16} - 1) * 4 * t_l) + 4*(t_l + t_m)$

=> $\mathbf{2^{18} t_l + 4t_m}$

4) Given:
Number of caches = 2 (L1& L2)
Capacity of L1 = C1
Capacity of L2 = C2 = 8 x C1
Block size of L1 = B1 =B
Block size of L2 = B2 = B
Hit time of L1 = t_1
Miss penalty of L1 = t_2
Miss penalty of L2 = t_m
Stride = 1
size = 8 x C2 = 64 x C1

For **stride = 1**, the algorithm reduces to having only one loop, the inner loop which iterates through every element in the array sequentially.

- Let us now consider only the case of the cache L1.

  The number of misses in L1 will be equivalent to the number of Lines required to store the array **A[size].** Since every element is accessed and accessed only once. Regardless of the cache size, access to every Line will cause a *cold-miss*.

  Hence the number of Misses in L1 =
  => Size of A / Size of a Block in L1
  => (8 x C2) / B1
  => **(64 x C1) / B**

  The number of Hits in L1 will be equivalent to:
  => The total size of the array – The number of Misses
  => (8 x C2) – ((64 x C1) / B)
  => (64 x C1) – ((64 x C1) / B)
  => **(64 x C1) (1- 1/B)**

- Now let us now consider only the case of the cache L2
  Now since both the caches are empty and the Block size of both the caches are the same, every miss in L1 will cause a miss in L2. (Since no extra Block is being stored in L2 for every miss in L1)
  Hence the number of Misses in L2 = Number of misses in L1
  => **(64 x C1) / B**
  From the above results, we can now calculate the total time of execution of the loop:
  => Total hit time in L1 + Total miss penalty for an L1 miss + Total miss penalty for an L2 miss + Total hit time for misses in L1
  => (t_l * Total no. of Hits in L1) + (t_2 * Total no. of Misses in L1) + (t_m * Total no. of Misses in L2) + (t_l * Total no. of Misses in L1)
  => (t_l * (64 * C1) (1- 1/B)) + (t_2 * (64 * C1) / B) + (t_m * (64 * C1) / B) + (t_l * (64 * C1)/B)
  => **64C1/B ($Bt_l - t_l + t_2 + t_m + t_l$)**
  => 64 C1 / B ($Bt_l + t_2 + t_m$)