Page 1157 Exercises 5.1, 5.2, 5.3, 5.5, 5.8, 5.18

5.1 In this exercise we look at memory locality properties of matrix computation. The following code is written in C, where elements within the same row are stored contiguously. Assume each word is a 32-bit integer. for (I=0; I<8; I++) for (J=0; J<8000; J++) A[I][J]=B[I][0]+A[J][I].

5.1.1 [5] <§5.1> How many 32-bit integers can be stored in a 16-byte cache block?

Cache size = 16 bytes.

Size of 1 integer = 4 bytes.

So number of integers that can be stored is (Cache size / integer size) = (16 / 4) = 4.

5.1.2 [5] <§5.1> Which variable references exhibit temporal locality?

I, J and B[I][0] exhibit temporal locality.

5.1.3 [5] <§5.1> Which variable references exhibit spatial locality? Locality is affected by both the reference order and data layout. The same computation can also be written below in Matlab, which differs from C by storing matrix elements within the same column contiguously in memory.

A[I][J] exhibits spatial locality.

5.1.4 [10] <§5.1> How many 16-byte cache blocks are needed to store all 32- bit matrix elements being referenced?

The matrix A has dimensions 8x8000.

Since each element of the matrix is 4 bytes.

Matrix size = 8 * 8000 * 4 = 256,000 bytes.

Cache blocks needed = 256,000 bytes / 16 bytes = 16,000 cache blocks.

5.1.5 [5] <§5.1> Which variable references exhibit temporal locality?

The 2D Array/Matrix A exhibits temporal locality as its being accessed multiple times over the lifetime of the loop.

5.1.6 [5] <§5.1> Which variable references exhibit spatial locality?

The 2D Array/Matrix A exhibits spatial locality as its elements are being accessed column by column then row by row in a sequential manner.

5.2 Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 32-bit memory address references, given as word addresses. 0x03, 0xb4, 0x2b, 0x02, 0xbf, 0x58, 0xbe, 0x0e, 0xb5, 0x2c, 0xba, 0xfd

5.2.1 [10] <§5.3> For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

0x03:

Binary address: 00000000 00000000 00000000 00000011

Tag: 00000000 00000000 00000000 0000 (first 28 bits)

Index: 0011 (last 4 bits)

Since it is a Empty cache, it's a miss.

0xb4:

Binary address: 00000000 00000000 00000000 10110100

Tag: 00000000 00000000 00000000 1011 (first 28 bits)

Index: 0100 (last 4 bits)

Since it is a Empty cache, it's a miss.

0x2b:

Binary address: 00000000 00000000 00000000 00101011

Tag: 00000000 00000000 00000000 0010 (first 28 bits)

Index: 1011 (last 4 bits)

Since it is a Empty cache, it's a miss.

0x02:

Binary address: 00000000 00000000 00000000 00000010

Tag: 00000000 00000000 00000000 0000 (first 28 bits)

Index: 0010 (last 4 bits)

Since it is a Empty cache, it's a miss.

0xbf:

Binary address: 00000000 00000000 00000000 10111111

Tag: 00000000 00000000 00000000 1011 (first 28 bits)

Index: 1111 (last 4 bits)

Since it is a Empty cache, it's a miss.

0x58:

Binary address: 00000000 00000000 00000000 01011000

Tag: 00000000 00000000 00000000 0101 (first 28 bits)

Index: 1000 (last 4 bits)

Since it is a Empty cache, it's a miss.

0xbe:

Binary address: 00000000 00000000 00000000 10111110

Tag: 00000000 00000000 00000000 1011 (first 28 bits)

Index: 1110 (last 4 bits)

Since it is a Empty cache, it's a miss.

0x0e:

Binary address: 00000000 00000000 00000000 00001110

Tag: 00000000 00000000 00000000 0000 (first 28 bits)

Index: 1110 (last 4 bits)

Since it is a Empty cache, it's a miss.

0xb5:

Binary address: 00000000 00000000 00000000 10110101

Tag: 00000000 00000000 00000000 1011 (first 28 bits)

Index: 0101 (last 4 bits)

Since it is a Empty cache, it's a miss.

0x2c:

Binary address: 00000000 00000000 00000000 00101100

Tag: 00000000 00000000 00000000 0010 (first 28 bits)

Index: 1100 (last 4 bits)

Since it is a Empty cache, it's a miss.

0xba:

Binary address: 00000000 00000000 00000000 10111010

Tag: 00000000 00000000 00000000 1011 (first 28 bits)

Index: 1010 (last 4 bits)

Since it is a Empty cache, it's a miss.

0xfd:

Binary address: 00000000 00000000 00000000 11111101

Tag: 00000000 00000000 00000000 1111 (first 28 bits)

Index: 1101 (last 4 bits)

Since it is a Empty cache, it's a miss.

5.2.2 [10] <§5.3> For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with two-word blocks and a total size of 8 blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

0x03:

Binary address: 00000000 00000000 00000000 00000011

Tag: 00000000 00000000 00000000 000000 (first 26 bits)

Index: 11 (last 2 bits)

Result: Cache Miss (Cache is initially empty)

0xb4:

Binary address: 00000000 00000000 00000000 10110100

Tag: 00000000 00000000 00000000 101101 (first 26 bits)

Index: 00 (last 2 bits)

Result: Cache Miss (Cache is initially empty)

0x2b:

Binary address: 00000000 00000000 00000000 00101011

Tag: 00000000 00000000 00000000 001010(first 26 bits)

Index: 11 (last 2 bits)

Result: Cache Miss (Cache is initially empty)

0x02:

Binary address: 00000000 00000000 00000000 00000010

Tag: 00000000 00000000 00000000 000000 (first 26 bits)

Index: 10 (last 2 bits)

Result: Cache Miss (Cache is initially empty)

0xbf:

Binary address: 00000000 00000000 00000000 10111111

Tag: 00000000 00000000 00000000 101111 (first 26 bits)

Index: 11(last 2 bits)

Result: Cache Miss (Cache is initially empty)

0x58:

Binary address: 00000000 00000000 00000000 01011000

Tag: 00000000 00000000 00000000 010110 (last 2 bits)

Index: 00 (last 2 bits)

Result: Cache Miss (Cache is initially empty)

0xbe:

Binary address: 00000000 00000000 00000000 10111110

Tag: 00000000 00000000 00000000 101111 (first 26 bits)

Index: 10 (last 2 bits)

Result: Cache Miss (Cache is initially empty)

0x0e:

Binary address: 00000000 00000000 00000000 00001110

Tag: 00000000 00000000 00000000 000011 (first 26 bits)

Index: 10 (last 2 bits)

Result: Cache Miss (Cache is initially empty)

0xb5:

Binary address: 00000000 00000000 00000000 10110101

Tag: 00000000 00000000 00000000 101101 (first 26 bits)

Index: 01 (last 2 bits)

Result: Cache Miss (Cache is initially empty)

0x2c:

Binary address: 00000000 00000000 00000000 00101100

Tag: 00000000 00000000 00000000 001011 (first 26 bits)

Index: 00 (last 2 bits)

Result: Cache Miss (Cache is initially empty)

0xba:

Binary address: 00000000 00000000 00000000 10111010

Tag: 00000000 00000000 00000000 101110 (first 26 bits)

Index: 10 (first 2 bits)

Result: Cache Miss (Cache is initially empty)

0xfd:

Binary address: 00000000 00000000 00000000 11111101

Tag: 00000000 00000000 00000000 111111(first 26 bits)

Index: 01 (first 2 bits)

Result: Cache Miss (Cache is initially empty)

5.2.3 [20] <§§5.3, 5.4> You are asked to optimize a cache design for the given references. There are three direct-mapped cache designs possible, all with a total of 8 words of data:
- C1 has 1-word blocks,
- C2 has 2-word blocks, and
- C3 has 4-word blocks.

- Cache 1 (C1):
  - Access Time: 2 cycles
  - Total Blocks: 8 words / 1 word per block = 8 blocks
  - Total References: 8 blocks * (1 read + 1 write) = 16 references
  - Miss Stall Time: 25 cycles
  - Misses: Miss Stall Time / Access Time = 25 cycles / 2 cycles = 12.5 misses (rounded up to 13 misses)

    - Miss Rate: (13 misses / 16 references) * 100 = 81.25%
- Cache 2 (C2):
    - Access Time: 3 cycles
    - Total Blocks: 8 words / 2 words per block = 4 blocks
    - Total References: 4 blocks * (1 read + 1 write) = 8 references
    - Miss Stall Time: 25 cycles
    - Misses: Miss Stall Time / Access Time = 25 cycles / 3 cycles = 8.33 misses (rounded up to 9 misses)
    - Miss Rate: (9 misses / 8 references) * 100 = 112.5%
- Cache 3 (C3):
    - Access Time: 5 cycles
    - Total Blocks: 8 words / 4 words per block = 2 blocks
    - Total References: 2 blocks * (1 read + 1 write) = 4 references
    - Miss Stall Time: 25 cycles
    - Misses: Miss Stall Time / Access Time = 25 cycles / 5 cycles = 5 misses
    - Miss Rate: (5 misses / 4 references) * 100 = 125%

Therefore, Cache 1 (C1) provides the best performance.

5.3 By convention, a cache is named according to the amount of data it contains (i.e., a 4 KiB cache can hold 4 KiB of data); however, caches also require SRAM to store metadata such as tags and valid bits. For this exercise, you will examine how a cache's configuration affects the total amount of SRAM needed to implement it, as well as the performance of the cache. For all parts, assume that the caches are byte addressable, and that addresses and words are 64 bits.

5.3.1. [10] <§5.3> Calculate the total number of bits required to implement a 32 KiB cache with two-word blocks.

Each word=8-bytes

Each block=2 words

Therefore each block contains 16=24 bytes

32 Kb Cache means = 2^15 bytes of data

Therefore

2^15/2^4 =2^11 lines of data that is blocks

Each 64bit address divided into:

A. 3bit offset
B. 1bit block offset
C. 11bit index (the reason is 211 lines)
D. 49 bit tag(64-3-1-11=49 bit tag)

The cache composed of =(2^15 x 8bits) +(2^11 x 49bits of tag) +(2^11*1 valid bits)= 364544 bits = 45568 bytes.

5.3.2. [10] <§5.3> Calculate the total number of bits required to implement a 64 KiB cache with 16-word blocks. How much bigger is this cache than the 32 KiB cache described in Exercise 5.3.1? (Notice that, by changing the block size, we doubled the amount of data without doubling the total size of the cache.)

- each block contains 128= 2^7 bytes

- 64 Kb Cache = 2^16 bytes
- 2^16/2^7 =2^9 lines of data blocks

Each 64bit address divided into:

A. 3bit offset

B. 4bit block offset

C. 9bit index (the reason is 29 lines)

D. 49 bit tag(64-3-4-9=48 bit tag)

The cache composed of

=(2^16 x 8bits of data) + (2^9 x 48bits of tag) +(2^9 x1 valid bits) = 549376 bits = 68672 bytes.

Thus it is 51% increase.

5.3.3. [5] <§5.3> Explain why this 64 KiB cache, despite its larger data size, might provide slower performance than the first cache.

- larger block size will lead to higher hit rate, and increased miss penalty.

5.5 For a direct-mapped cache design with a 32-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
31–10	9–5	4–0

5.5.1 [5] <§5.3> What is the cache block size (in words)?

cache block size = 25 bytes = 25 / 22 = 23 words = 8 words.

5.5.2 [5] <§5.3> How many entries does the cache have?

number of entries in the cache = 2^5 = 32 entries.

5.5.3 [5] <§5.3> What is the ratio between total bits required for such a cache implementation over the data storage bits?

data bits = 256 bits.

tag bits = 22 bits

valid bits = 1 bit.

total bits needed to implement the cache per block = 256 + 22 + 1 = 279 bits.

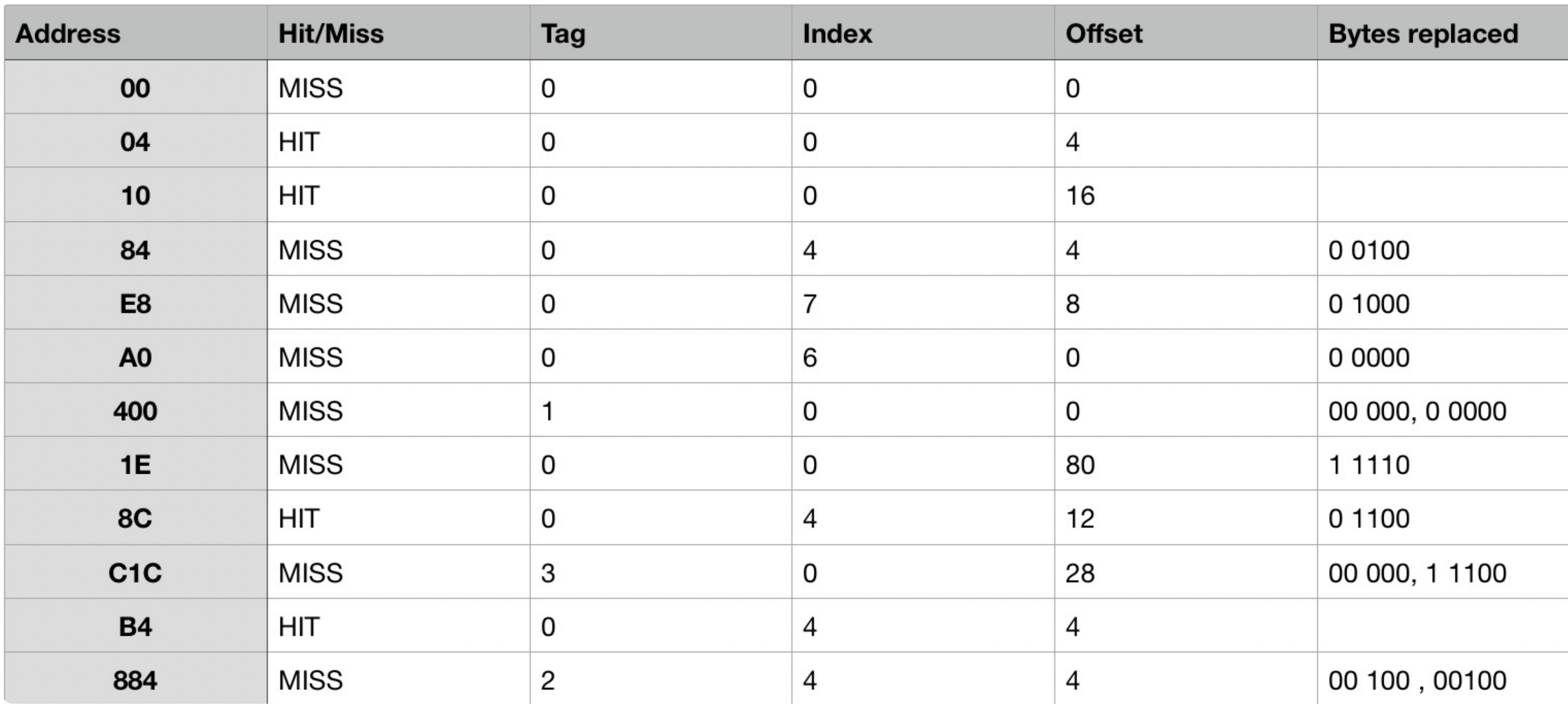
Ratio = total bits / data bits = 279 / 256 = 1.089

Beginning from power on, the following byte-addressed cache references are recorded. L2 Write back, write allocate

Address:

Hex 00	**04**	**10**	**84**	**E8**	**A0**	**400**	**1E**	**8C**	**C1C**	**B4**	**884**
Dec 0	**4**	**16**	**132**	**232**	**160**	**1024**	**30**	**140**	**3100**	**180**	**2180**

5.5.4 [20] <§5.3> For each reference, list (1) its tag, index, and offset, (2) whether it is a hit or a miss, and (3) which bytes were replaced (if any).



Address	Hit/Miss	Tag	Index	Offset	Bytes replaced
00	MISS	0	0	0	
04	HIT	0	0	4	
10	HIT	0	0	16	
84	MISS	0	4	4	0 0100
E8	MISS	0	7	8	0 1000
A0	MISS	0	6	0	0 0000
400	MISS	1	0	0	00 000, 0 0000
1E	MISS	0	0	80	1 1110
8C	HIT	0	4	12	0 1100
C1C	MISS	3	0	28	00 000, 1 1100
B4	HIT	0	4	4	
884	MISS	2	4	4	00 100 , 00100

5.5.5 [10] <§5.3> What is the hit ratio?

Hit ratio = 4/13 = 0.307

5.5.6 [20] <§5.3> List the final state of the cache, with each valid entry represented as a record of <index, tag, data>.

- <4,2, mem[0x880] - mem[0x89F]>
- <0,3, mem[0xC00]-mem[0xC1F]>
- <5,0, mem[0xA0]-mem[0xBF]>
- <7, 0, mem[0xE0] - mem[FF]>

5.8 Media applications that play audio or video files are part of a class of workloads called "streaming" workloads (i.e., they bring in large amounts of data but do not reuse much of it). Consider a video streaming workload that accesses a 512 KiB working set sequentially with the following address stream:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9 …

5.8.1 [10] <§§5.4, 5.8> Assume a 64 KiB direct-mapped cache with a 32-byte block. What is the miss rate for the address stream above? How is this miss rate sensitive to the size of the cache or the working set? How would you categorize the misses this workload is experiencing, based on the 3C model?

- The miss rate = 512kb/32b = 16K misses.
-

5.8.2 [5] <§§5.1, 5.8> Re-compute the miss rate when the cache block size is 16 bytes, 64 bytes, and 128 bytes. What kind of locality is this workload exploiting?

- No. of misses In 16byte, 512K/16 = 32K.
- Hence miss rate = 12.5%
- No. of misses in 64byte, 512K/64 = 8K.
- Hence miss rate = 3.125%
- No. of misses in 128byte, 512K/128= 4K.
- Hence miss rate = 1.5625%

5.8.3 [10] <§5.13 > "Prefetching" is a technique that leverages predictable address patterns to speculatively bring in additional cache blocks when a particular cache block is accessed.
One example of prefetching is a stream Buffer that prefetches sequentially adjacent cache blocks into a separate buffer when a particular cache block is brought in. If the data is found in the prefetch buffer, it is considered as a hit and moved into the cache and the next cache block is prefetched. Assume a two-entry stream buffer and assume that the cache latency is such that a cache block can be loaded before the computation on the previous cache block is completed. What is the miss rate for the address stream above?

The miss rate will approximately be near 0.

5.18 In this exercise, we will examine space/time optimizations for page tables. The following list provides parameters of a virtual memory system.

Virtual Address	Physical DRAM	Page Size	PTE Size (byte)
43	16 GB	4 KB	4

5.18.1 [10] <§5.7> For a single-level page table, how many page table entries (PTEs) are needed? How much physical memory is needed for storing the page table?

- Virtual address bit = 43 bit,
- virtual address memory size = 2^{43}
- Page Size = 4 KB = 2^{12}
- PTE needed = 2^{43} / 2^{12} = 2^{31}
- Now each entry size = PTE Size = 4 = 2^{2}
- Physical Memory needed = 2^{2}*2^{31} = 2^{33}

5.18.2 [10] <§5.7> Using a multilevel page table can reduce the physical memory consumption of page tables, by only keeping active PTEs in physical memory. How many levels of page tables will be needed if the segment tables (the upper-level page tables) are allowed to be of unlimited size? And how many memory references are needed for address translation if missing in TLB?

- Only two levels of page tables are needed: segment table and a page table.
- Two memory references are needed for address translation if missing in TLB: one to access the segment table and another to access the page table.

5.18.3 [10] <§5.7> Suppose the segments are limited to the 4 KB page size (so that they can be paged). Is 4 bytes large enough for all page table entries (including those in the segment tables?

Yes, because each PTE only needs to store a physical page number, which can fit within 4 bytes given that we have 16GB of physical memory and a page size of 4KiB.

5.18.4 [10] <§5.7> How many levels of page tables are needed if the segments are limited to the 4 KiB page size?

5.18.5 [15] <§5.7> An inverted page table can be used to further optimize space and time. How many PTEs are needed to store the page table? Assuming a hash table implementation, what are the common case and worst case numbers of memory references needed for servicing a TLB miss?

- The number of page table entries (PTEs) needed would be equal to the number of page frames in the system.
- the common case number of memory references for servicing a TLB miss would be the average number of hash table lookups required to find the desired entry.
- The worst case number of memory references would occur when there are hash collisions, leading to additional probing or chaining operations to resolve the collision.